GRAPES: Learning to Sample Graphs for Scalable Graph Neural Networks

Taraneh Younesian *t.younesian@vu.nl*
Vrije Universiteit Amsterdam

Daniel Daza *d.f.dazacruz@amsterdamumc.nl*
Amsterdam UMC

Emile van Krieken *Emile.van.Krieken@ed.ac.uk*
University of Edinburgh

Thiviyan Thanapalasingam *t.singam@uva.nl*
University of Amsterdam

Peter Bloem *p.bloem@vu.nl*
Vrije Universiteit Amsterdam

Reviewed on OpenReview: *https://openreview.net/forum?id=QIOl842vSq*

Abstract

Graph neural networks (GNNs) learn to represent nodes by aggregating information from their neighbors. As GNNs increase in depth, their receptive field grows exponentially, leading to high memory costs. Several works in the literature proposed to address this shortcoming by sampling subgraphs or by using historical embeddings. These methods have mostly focused on benchmarks of single-label node classification on *homophilous graphs*, where neighboring nodes often share the same label. However, most of these methods rely on static heuristics that may not generalize across different graphs or tasks. We argue that the sampling method should be *adaptive*, adjusting to the complex structural properties of each graph. To this end, we introduce GRAPES, an adaptive sampling method that learns to identify the set of nodes crucial for training a GNN. GRAPES trains a second GNN to predict node sampling probabilities by optimizing the downstream task objective. We evaluate GRAPES on various node classification benchmarks involving homophilous as well as heterophilous graphs. We demonstrate GRAPES' effectiveness in accuracy and scalability, particularly in multi-label heterophilous graphs. Additionally, GRAPES uses orders of magnitude less GPU memory than a strong baseline based on historical embeddings. Unlike other sampling methods, GRAPES maintains high accuracy even with smaller sample sizes and, therefore, can scale to massive graphs. Our implementation is publicly available online.[1]

1 Introduction

Despite the broad range of applications of GNNs (Nettleton, 2013; Wu et al., 2022; Li et al., 2022; Kipf & Welling, 2016; Velickovic et al., 2017; Yun et al., 2019), scalability remains a significant challenge (Serafini & Guan, 2021). Unlike traditional machine learning problems, where data is assumed to be i.i.d., the graph structure introduces dependencies between a node and its neighborhood. This complicates partitioning data into mini-batches. Additionally, the number of nodes that a GNN needs to process increases exponentially with the number of layers (Ma et al., 2025).

[1]Available at https://github.com/dfdazac/grapes.

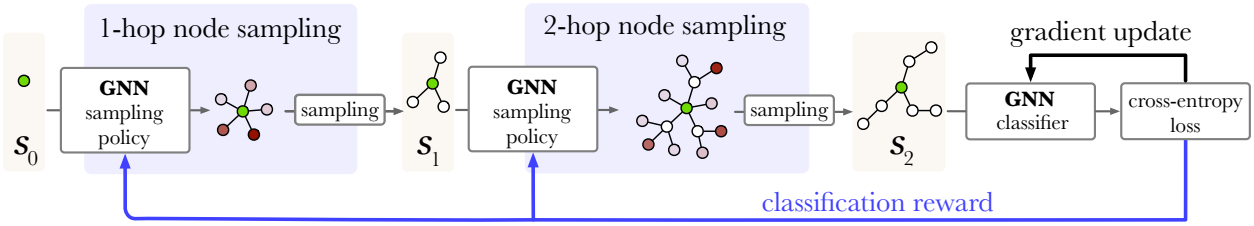

Figure 1: Overview of GRAPES. First, GRAPES processes a target node (green) by computing node inclusion probabilities on its 1-hop neighbors (shown by node color shade) with a sampling GNN. Given these probabilities, GRAPES samples k nodes. Then, GRAPES repeats this process over nodes in the 2-hop neighborhood. We pass the sampled subgraph to the classifier GNN for target node classification. Finally, GRAPES uses the classification loss to update the classifier GNN and to reward the sampler GNN.

Most current graph sampling methods use a fixed heuristic to compute node inclusion probabilities: The sampling process is independent of the node features, graph structure, or the GNN task performance (Chen et al., 2018b; Zou et al., 2019; Zeng et al., 2019). For simple graphs like *homophilous graphs* (Zhu et al., 2020; Zheng et al., 2022; Zhao et al., 2023) where there is a strong correlation between labels of a node and its neighbors, a fixed heuristic typically suffices. However, these methods fall short when applied to graphs in which only a few of the nodes in the neighborhood provide relevant information for the task at hand. Consequently, we study *adaptive sampling*, where the sampling method adapts dynamically to the task by learning which nodes should be included. Our adaptive sampling technique learns to sample by directly minimizing the classification loss of the downstream GNN. This differs from most existing graph sampling methods, which aim to approximate the full-batch GNN (Chen et al., 2018b; Zou et al., 2019; Zeng et al., 2019; Huang et al., 2018), opting for an indirect approach to achieve high accuracy.

We introduce a straightforward adaptive sampling method called **Gr**aph **A**da**p**tiv**e** **S**ampling (GRAPES). As illustrated in Figure 1, GRAPES samples a subgraph around the target nodes in a series of steps. At each step, a *sampling policy* GNN computes inclusion probabilities for the nodes neighboring the current subgraph, which the sampler uses to select a subset. Once the sampling is complete, the resulting subgraph is passed to a second *classifier* GNN for classification. The classification loss is then backpropagated to train both GNNs.

To enable backpropagation through the sampling step, we need a gradient estimator. We compare a reinforcement learning (RL) approach and a GFlowNet (GFN) approach (Bengio et al., 2021a). Both approaches allow us to consider the sampling process and the GNN computation together and to train them concurrently. This allows the sampler to adapt based on factors like the node features, the graph structure, the sample size, and other contextual features.

In multi-class classification– where a node can be assigned only one of multiple classes– on homophilous graphs, even a simple strategy like random sampling could be sufficient, since the neighborhood of a node contains nodes with similar labels (see Fig. 2a). In the case of multi-label classification, on the other hand, the number of possible labels assigned to a node is $|2^{\mathcal{Y}}|$, where $\mathcal{Y}$ is the unique set of labels. Furthermore, in heterophilous graphs, the relationship between the neighborhood of a node and its label is more complex (see Fig. 2b). We argue that in such cases, a sampling policy that directly adapts to the properties of the graph and the downstream task is crucial.

Therefore, in addition to the datasets commonly used in the literature, we evaluate GRAPES on several heterophilous and/or multi-label graphs and demonstrate its effectiveness on these complicated graphs. Moreover, we spend significant effort on the evaluation protocol for fairly testing graph sampling methods. In particular, we ensure that all sampling methods are evaluated under the same conditions and on the same GNN architecture to eliminate any confounding factors and to ensure that any changes in performance can only be attributed to the sampling method. To the best of our knowledge, we are the first to perform such a rigorous comparison on twelve varied datasets.

(a) (b)

Figure 2: The downstream task and the structural properties of the graph affect the strategy used for sampling. Fig. 2a shows a citation network where articles citing each other are likely to belong to the same category. Fig. 2b shows a graph of interacting proteins, each with different biological functions.

Additionally, we provide a theoretical analysis indicating the effectiveness of adaptive sampling on a specific category of heterophilous graphs, where only a subset of neighbors contains relevant information for an accurate node classification.

We evaluate GRAPES on several node classification tasks and find that

1. **Performance.** GRAPES achieves state-of-the-art performance in multi-label classification benchmarks over heterophilous graphs, while performing competitively on multi-class classification on homophilous graphs.
2. **Memory Efficiency.** GRAPES uses orders of magnitude less GPU memory when compared with a strong baseline that relies on historical embeddings for scaling GNN training.
3. **Robustness.** In comparison with other methods, GRAPES maintains good performance under exponential reductions in sample size.

2 Related Work

Fixed Sampling Policy: In this category, the sampling policy is independent of the training of the GNN and is based on a fixed heuristic that does not involve any training. Given a set of target nodes, *i.e.* the nodes that are to be classified, node-wise sampling methods sample a given number of nodes for each target node. GraphSage (Hamilton et al., 2017) is a node-wise sampling method that randomly samples nodes. However, node-wise sampling can result in nodes being sampled multiple times redundantly because they can be the neighbors of several nodes (Zou et al., 2019). A more efficient approach is layer-wise sampling, for example, FastGCN (Chen et al., 2018b) and LADIES (Zou et al., 2019). They aim to minimize variance by sampling nodes in each layer with probabilities proportional to their degree. MVS-GNN (Cong et al., 2020) proposes a fixed sampling policy with the aim of minimizing sampling variance by decoupling it into two components. The first component, embedding approximation variance, arises from neighbor sampling and is mitigated through the use of historical embeddings. The second component, stochastic gradient variance, results from mini-batching and is addressed by incorporating the norm of the node gradients into the sampling process. Moreover, some techniques focus on sampling subgraphs in each mini-batch, like GraphSAINT (Zeng et al., 2019) and ClusterGCN (Chiang et al., 2019). While these techniques effectively scale GNNs to larger graphs, they do not adapt to the sampling policy based on the GNN's performance on the task. In the graph signal processing community, the authors in (Geng et al., 2023) propose a node sampling technique based on (Anis et al., 2016) to sample nodes for a unique and stable graph signal reconstruction. However, this method is only applied to small graphs.

Learnable Sampling Policy: A few methods learn the probability of including a node based on feedback from the GNN. AS-GCN (Huang et al., 2018) is a method that learns a linear function that estimates the node probabilities layer-wise. Similarly, PASS (Yoon et al., 2021) learns a mixture of a random distribution and a learned policy with RL. Like GRAPES, it is task-adaptive; however, its sampling probability uses a

bilinear similarity measure between neighboring node features, meaning the graph structure is not considered. FairSample Cong et al. (2023) is a learnable sampling policy that borrows the PASS training mechanism that utilizes reinforcement learning to combine classification loss and a fairness loss to balance accuracy and fairness towards the underrepresented nodes. GNN-BS (Liu et al., 2020) formulates the node-wise sampling problem as a bandit problem and updates the sampling policy according to a reward function that reduces the sampling variance. SubMix (Abu-El-Haija et al., 2023) proposes a mixture distribution of sampling heuristics with learnable mixture weights. DSKReG (Wang et al., 2021) learns the relevance of items in a user-item knowledge graph by jointly optimizing the sampling strategy and the recommender model. The majority of these methods focus on variance reduction and fail to consider the classification loss, unlike GRAPES. We argue that adaptivity to the classification loss allows for sampling the influential nodes depending on the task and results in better performance.

Other Scalable Methods: Authors of (You et al., 2022; Zhang et al., 2024) use graph lottery tickets to eliminate unnecessary edges, while DSpar (Liu et al., 2023) uses a simple degree-based heuristic to sparsify the graph. Ruiz et al. (2023) proposes a method to transfer the weights of a GNN trained on a mid-sized graph to larger graphs, given the graphon similarity between the graphs. Another group of papers uses historical embeddings of the nodes when updating the target nodes' embeddings (Chen et al., 2018a; Fey et al., 2021; Yu et al., 2022; Shi et al., 2023). GAS (Fey et al., 2021) approximates the embeddings of the 1-hop neighbors using the historical embeddings of those nodes learned in the previous training iterations. These methods reduce the GPU memory usage by training in mini-batches and learning from the 1-hop neighbors with the historical embeddings saved in CPU memory. Unlike GRAPES, they process the whole graph.

3 Background: GNN Training and Sampling

We first provide the necessary background about GNNs and graph sampling. Although our method is independent of the choice of GNN architecture, we limit our discussion to the GCN architecture for simplicity (Kipf & Welling, 2016).

Let $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ be an undirected graph with a list of N nodes $\mathcal{V} = \{1, \ldots, N\}$ and a set of edges $\mathcal{E}$. The adjacency matrix $A \in \{0,1\}^{N \times N}$ indicates a connection between a pair of nodes. Next, let $\hat{A} = \tilde{D}^{-1/2}\tilde{A}\tilde{D}^{-1/2}$, where $\tilde{A} = A + I$ and where $\tilde{D}$ is the degree matrix of $\tilde{A}$. Let $X \in \mathbb{R}^{N \times f}$ be the node embeddings and let Y be the labels for the *target nodes* $\mathcal{V}^t \subset \mathcal{V}$, where $\mathcal{V}^t$ indexes the nodes with a label.

We consider a GCN with L layers. The output of the l-th layer of the GCN is $H^{(l)} = \sigma(\tilde{A}H^{(l-1)}W^{(l)})$, where $W^{(l)}$ is the weight matrix of GCN layer l and σ is a non-linear activation function. For a node $i \in \mathcal{V}$, this corresponds to the update

$$h_i^{(l)} = \sigma\left(\sum_{j \in \mathcal{N}(i) \cup \{i\}} \hat{A}_{ij} h_j^{(l-1)} W^{(l)}\right), \tag{1}$$

where $\mathcal{N}(i) : \mathcal{V} \to 2^{\mathcal{V}}$ is the set of i's neighbors excluding i.

As the number of layers increases, the computation of the embedding of the node i involves neighbors from further hops. As a result, the neighborhood size grows rapidly with the number of layers. We study how to sample the graph to overcome this growth. We focus on layer-wise sampling, a common type of graph sampling approach (Chen et al., 2018b; Zou et al., 2019; Huang et al., 2018). First, we divide the target nodes into mini-batches of size b. Then, in each layer, we sample k nodes $\mathcal{V}^{(l)}$ among the neighbors of the nodes in the previous layer using the sampling policy q. To make this precise, we will need some additional notation. The computation of the output of layer l is then:

$$\begin{aligned} \text{for all } i \in K^{(l)}, \quad h_i'^{(l)} &= \sigma\left(\sum_{j \in K^{(l-1)}} \hat{A}_{ij}^{(l)} h_j'^{(l-1)} W^{(l)}\right) \\ \text{with} \quad \mathcal{V}^{(l-1)}, \mathcal{V}^{(l)} &\sim q(\mathcal{V}^{(l-1)}, \mathcal{V}^{(l)} | k), \end{aligned} \tag{2}$$

where

1. $K^{(0)} = \mathcal{V}^{(0)}$ is the set of target nodes in the current mini-batch, where $\mathcal{V}^{(0)} \subset \mathcal{V}^t$;
2. $K^{(l)} = \mathcal{V}^{(l)} \cup \mathcal{V}^{(0)}$ for all $l \in 1, \ldots, L$ adds the batch nodes $\mathcal{V}^{(0)}$ to the sampled nodes $\mathcal{V}^{(l)}$ to ensure self-loops between the batch nodes;
3. $\mathcal{V}^{(l)} \subseteq \mathcal{N}(K^{(l-1)})$ are the nodes sampled in layer l among the neighbors $\mathcal{N}(K^{(l-1)})$ of the nodes in $K^{(l-1)}$. Note that $\mathcal{V}^{(l)}$ cannot contain nodes in $K^{(l-1)}$;
4. $\hat{A}^{(l)} = {D^{(l)}}^{-1/2} A^{(l)} {D^{(l)}}^{-1/2}$ is computed from the adjacency matrix $A^{(l)}$ containing the edges between $K^{(l)}$ and $K^{(l-1)}$, and the corresponding degree matrix $D^{(l)}$.[2] To be precise, the entries $A^{(l)}_{ij}$ are 1 if and only if $i \in K^{(l)}$, $j \in K^{(l-1)}$ and if there is an edge (i, j) in the original graph, that is, $\tilde{A}_{ij} = 1$;
5. q is a sampling policy that samples the k nodes $\mathcal{V}^{(l)}$.

Existing layer-wise sampling methods, such as LADIES or FastGCN, use a fixed heuristic to determine q, for instance, by computing node probabilities proportional to the node degrees. However, an adaptive method *learns* the distribution q instead.

4 Graph Adaptive Neighbor Sampling (GRAPES)

We introduce Graph Adaptive Sampling (GRAPES). GRAPES is a layer-wise and layer-dependent adaptive sampling method that learns a sampling policy that minimizes the training objective conditioned on the input graph. In each layer l, we sample a subset $\mathcal{V}^{(l)}$ that is much smaller than the neighborhood of the previous layer. That is, $|\mathcal{V}^{(l)}| = k \ll |\mathcal{N}(K^{(l-1)})|$, where again $K^{(l-1)} = \mathcal{V}^{(0)} \cup \mathcal{V}^{(l-1)}$ adds the batch nodes to the sampled nodes of the previous layer. We use a second GNN to compute the inclusion probability for each node in $\mathcal{N}(K^{(l-1)})$. In the remainder of this section, we describe our sampling policy q and training methods for the sampling policy. Furthermore, Algorithm 1 shows one epoch of GRAPES in pseudocode.

4.1 Sampling policy

Next, we specify the sampling policy GNN $\mathrm{GCN}_\mathrm{S}(K^{(l-1)})$, which computes inclusion probabilities on the subgraph created from the nodes in $K^{(l-1)}$ and their neighbors $\mathcal{N}^{(l-1)}$. The sampling policy q decomposes as $q(\mathcal{V}^{(1)}, \ldots, \mathcal{V}^{(L)} | \mathcal{V}^{(0)}) = \prod_{l=1}^{L} q(\mathcal{V}^{(l)} | \mathcal{V}^{(0)}, \ldots, \mathcal{V}^{(l-1)})$. We compute each factor $q(\mathcal{V}^{(l)} | \mathcal{V}^{(0)}, \ldots, \mathcal{V}^{(l-1)})$ as a product of Bernoulli inclusion probabilities, given by GCN_S, for each node i in the neighborhood $\mathcal{N}(K^{(l-1)})$:

$$q(\mathcal{V}^{(l)} | \mathcal{V}^{(0)}, \ldots, \mathcal{V}^{(l-1)}) = \prod_{i \in \mathcal{N}(K^{(l-1)})} \mathrm{Bern}(i \in \mathcal{V}^{(l)} | p_i), \quad p_i = \mathrm{GCN}_\mathrm{S}(K^{(l-1)})_i. \tag{3}$$

In addition to the regular embeddings X, the sampler GNN also has access to a one-hot vector of length $L + 1$ that records the value $l + 1$ for nodes sampled in layer l (with 1 recorded for the target nodes). This allows the sampler to differentiate between nodes sampled in different layers.

4.2 Sampling exactly k nodes

We have a clear constraint on the number of nodes we want to include in training: In Equation 2, we sample exactly k nodes without replacement. However, our sampling policy $q(\mathcal{V}^{(l)} | \mathcal{V}^{(0)}, \ldots, \mathcal{V}^{(l-1)})$ consists of many independent Bernoulli distributions, and it is highly unlikely that we sample exactly k nodes from this distribution.

Instead, we use the Gumbel-Top-k trick (Vieira, 2014; Huijben et al., 2022), which selects a set of exactly k nodes $\mathcal{V}^{(l)}$ by perturbing the log probabilities randomly and taking the top-k among those:

$$\mathcal{V}^{(l)} = \underset{i \in \mathcal{N}(K^{(l-1)})}{\text{top-k}} \log p_i + \epsilon_i, \quad \epsilon_i \sim \mathrm{Gumbel}(0, 1) \tag{4}$$

This guarantees a sample from $q(\mathcal{V}^{(l)} | \mathcal{V}^{(0)}, \ldots, \mathcal{V}^{(l-1)}, k)$ that conditions on the number of nodes sampled.

[2] Unlike the full-batch GCN, the adjacency matrix varies across layers when sampling because each layer involves a different set of nodes. Note also that, unlike the full-batch setting, message passing is *asymmetric*: node i may be updated from node j but not vice versa.

While this results in a tractable and adaptive sampling procedure of exactly k nodes for the classifier GNN, we have not yet given a method for learning the sampling policy. Unfortunately, a sampling operation, or in this case, the top-k operation, provides no functional gradient (Mohamed et al., 2020). We will resort to simple methods from the reinforcement learning and GFlowNet literature to still be able to train the sampling policy, which we will explain next.

The choice of a Bernoulli distribution for sampling node neighborhoods is motivated by the idea that when conditioned on a target node, we can sample a node independent of other nodes in the neighborhood to maintain efficiency. Other assumptions could be more elaborate, such as sampling jointly pairs of nodes at a time, but this would in turn increase the computational cost of computing probabilities and sampling. On the other hand, the Gumbel top-k trick allows us to sample exactly k elements without replacement, and while there are other applicable methods such as reservoir sampling, they are mathematically equivalent (Huijben et al., 2022).

4.3 Training the Sampling Policy

We train the sampling policy to minimize the classification loss $\mathcal{L}_{\text{C}}$ of the classifier GNN[3]. We use two methods to train the sampling policy GCN_{S}: a reinforcement learning (RL) method and a GFlowNet (GFN) method. We also experimented with the straight-through estimator (Bengio et al., 2013) for learning in GRAPES, which we observed not to perform well due to increased computation and high bias in the estimated gradients (see Appendix G).

REINFORCE (GRAPES-RL) In the REINFORCE-based method, we use a simple REINFORCE estimator (Williams, 1992) to compute an unbiased gradient of the classification loss.

$$\mathcal{L}_{\text{RL}}(X, Y, \mathcal{V}^{(0)}) = \mathcal{L}_{\text{C}}(X, Y, K^{(0)}, \dots, K^{(L)}) \log q(\mathcal{V}^{(1)}, \dots, \mathcal{V}^{(L)} | \mathcal{V}^{(0)}), \tag{5}$$

where we sample $\mathcal{V}^{(1)}, \dots, \mathcal{V}^{(L)} \sim q(\mathcal{V}^{(1)}, \dots, \mathcal{V}^{(L)} | \mathcal{V}^{(0)}, k)$. Taking the derivative with respect to the parameters of q results in the standard REINFORCE estimator. Note that this is an off-policy estimator since we sample from q conditioned on the number of samples k with the Gumbel-Top-k trick, but compute gradients with respect to the distribution unconditioned on k. This is because computing likelihoods conditioned on k, although possible (Ahmed et al., 2023), is computationally expensive. We discuss this issue in more detail in Appendix A.

GFlowNets (GRAPES-GFN) The second method uses the Trajectory Balance loss (Malkin et al., 2022a) from the GFlowNet literature (Bengio et al., 2021b;a), which is known to perform well in off-policy settings (Malkin et al., 2022b).

$$\begin{aligned} \mathcal{L}_{\text{GFN}}(X, Y, \mathcal{V}^{(0)}) = \Big(& \log Z(\mathcal{V}^{(0)}) + \log q(\mathcal{V}^{(1)}, \dots, \mathcal{V}^{(L)} | \mathcal{V}^{(0)}) \\ & + \alpha \cdot \mathcal{L}_{\text{C}}(X, Y, K^{(0)}, \dots, K^{(L)}) \Big)^2, \end{aligned} \tag{6}$$

where $\log Z(\mathcal{V}^{(0)})$ is a small GCN that predicts a scalar from the target nodes, and α is a tunable reward scaling hyperparameter. For a detailed derivation, see Appendix F.3. GFlowNets minimize an objective that ensures sampling in proportion to the negative classification loss, rather than minimizing it like in REINFORCE. This may have benefits in the exploratory behavior of the sampler, as it encourages the training of diverse sets of nodes, instead of only the single best set of nodes.

Memory complexity. Layer-wise sampling methods like GRAPES, FastGCN, and LADIES have a sampling space complexity of $O(Dbk)$, where D is the maximum node degree, b is the batch size, k is the sample size, and L is the number of layers. In GAS time complexity is $O(bk)$ due to the use of historical embeddings, and memory complexity is $O(DbL + N)$ where N is the extra overhead for storing historical embeddings for the N nodes in the graph, which can be significantly larger than DbL.

[3]GRAPES can be extended to other tasks, but we focus on node classification in the current work.

Algorithm 1 One GRAPES epoch

Require: Graph $\mathcal{G}$, node embeddings X, node labels Y, target nodes $\mathcal{V}^t$, batch size b, sample size k, GCN for classification GCN_C, and GCN for sampling policy GCN_S.

Divide target nodes $\mathcal{V}^t$ into batches $\mathcal{V}^{(0)}$ of size b
for each batch $\mathcal{V}^{(0)}$ **do**
  $K^{(0)} \leftarrow \mathcal{V}^{(0)}$
  for layer $l = 1$ to L **do**
    for node i in $\mathcal{N}(K^{(l-1)})$ **do**
      $p_i \leftarrow \text{GCN}_S(K^{(l-1)})_i$ ▷ Compute probabilities of inclusion
      $\epsilon_i \sim \text{Gumbel}(0, 1)$ ▷ Sample Gumbel noise
    $\mathcal{V}^{(l)} \leftarrow \text{top-k}_{i \in \mathcal{N}(K^{(l-1)})} \log p_i + \epsilon_i$ ▷ Get k best nodes (Eq. 4)
    $K^{(l)} \leftarrow \mathcal{V}^{(0)} \cup \mathcal{V}^{(l)}$ ▷ Add target nodes
  $\ell_\text{C} \leftarrow \mathcal{L}_\text{C}(X, Y, K^{(0)}, \ldots, K^{(L)})$ ▷ Compute classification loss
  Compute sampling policy loss ℓ_S from Eq. 5 or 6
  Update parameters of GCN_S by minimizing ℓ_S
  Update parameters of GCN_C by minimizing ℓ_C

5 Theoretical Analysis

In this section, we provide theoretical insights on the performance difference between adaptive and non-adaptive samplers, and we prove that adaptive sampling can achieve higher accuracy than non-adaptive sampling for a specific category of heterophilous graphs. We show that if there exists specific crucial information among the neighbors, there are adaptive sampling methods that perfectly identify these neighbors, while non-adaptive sampling methods cannot distinguish them from the other neighbors.

Let $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ be an undirected graph with a set of N nodes $\mathcal{V} = \{1, \ldots, N\}$ and a set of edges $\mathcal{E} = \{e_{ij}\}_{i,j=1}^{N}$ where e_{ij} denotes an edge between v_i and v_j. Let $X \in \mathbb{R}^{N \times F}$ be the node features and let $Y = \{y_i\}_{i=1}^{N}$ be a set of node labels. The adjacency matrix $A \in \{0, 1\}^{N \times N}$ indicates a connection between a pair of nodes. Let $\mathcal{N}(v_i) = \{v_j | e_{ij} \in \mathcal{E}\}$ indicate the set of v_i's neighbors and $\mathcal{N}^L(v_i) = \{v_j | l(v_i, v_j) \leq L\}$ indicate the L-hop neighborhood of v_i, where $l(v_i, v_j)$ indicate the shortest path between v_i and v_j. In the following definitions and theorems, for simplicity, we consider the batch size of one.

Definition 1 (Neigbor sampling)**.** *A* neighbor sampling method *is a method that assigns scores to neighbors of a target node, and for a given sampling budget K, samples K neighbors proportional to their score. An L-layer neighbor sampling method, for each layer l, assigns scores to the l-hop neighbors of a target node.*

Definition 2 (Featureless and feature-based sampling)**.** *An* L-layer feature-less sampling *method is a neighbor sampling method in which the sampling score for each node v_i is a function of only its L-hop neighborhood, i.e. $s(v_i) = g(v_i, \mathcal{N}^L(v_i))$, where $s(v_i)$ is the sampling for node v_i and g is the sampling function. In an* L-layer feature-based sampling *method, the sampling function is a function of the node features in addition to the L-hop neighborhood, that is, $s(v_i) = g'(v_i, X, \mathcal{N}^L(v_i))$.*

In practice, all feature-based sampling methods are adaptive since the sampling methods learns the sampling score from the features, and all non-adaptive methods are featureless. Therefore, in this section, we refer to the sampling methods as adaptive and non-adaptive.

Random sampling is a non-adaptive sampler with a constant sampling score. LADIES and FastGCN are non-adaptive samplers whose sampling scores are proportional to the node degrees. GraphSAINT-RW, i.e., the random walk-based sampler of GraphSAINT, which we used in our experiments, is a non-adaptive sampler where the score correlates with how frequently a node appears in L-hop neighborhoods, influenced by both the node's degree and its position in the graph. AS-GCN, PASS, and GRAPES are adaptive samplers.

Definition 3 (Zhu et al. (2020))**.** *Edge homophily is the ratio of edges that connect two nodes of the same label:*

$$h = \frac{|\{e_{ij} \in \mathcal{E} : y_i = y_j\}|}{|\mathcal{E}|}$$

In the next theorem, we show that the optimal adaptive and non-adaptive samplers perform differently for a certain category of graphs, with adaptive samplers perfectly sampling the influential neighbors and non-adaptive samplers sampling such nodes with a vanishing probability.

Theorem 1. *There exist undirected graphs such that a GCN with an L-layer adaptive sampler and a sampling rate of K neighbors per node per layer can perform with perfect accuracy, while GCNs with a non-adaptive sampler can achieve an accuracy higher than chance level only with probability $p \leq \frac{LK}{N}$.*

For the proof of this theorem, please refer to Appendix H. In the proof, we construct a family of distributions over graphs where the presence of a single neighbor is essential for a correct classification of a target node. Then, we show that only feature-based samplers, by learning from the node features, can sample those neighbors, while featureless samplers, with overwhelming probability, cannot. Since, in practice, all feature-based samplers are adaptive and, inversely, all non-adaptive samplers are featureless, we conclude that adaptive models can solve this task, while non-adaptive methods cannot.

Additionally, we show that such graphs are heterophilous. Moreover, often, multi-label graphs are heterophilous because neighboring nodes are less likely to share the exact same set of labels. Even slight variations in the labels of neighboring nodes can lead to heterophily. Therefore, adaptive sampling methods outperform non-adaptive ones in such graphs.

6 Experiments

Our experiments aim at answering the following research question: *given a fixed sampling budget and GNN architecture, what is the effect of training with an adaptive policy for layer-wise sampling, in comparison with the related work?* While several works in the literature of sampling for GNNs have focused on classification benchmarks to demonstrate the performance of sampling algorithms, several confounding factors in their experimental setup prevent a proper understanding of whether the perceived performance improvements result from the sampling method itself. Examples of such confounding factors found in related work include using different architectures, like the GCN (Kipf & Welling, 2016) or GAT (Velickovic et al., 2017), different sizes for the hidden layers, number of layers, batch sizes, number of nodes sampled per layer, and the number of training epochs. These are factors independent of sampling algorithms that nonetheless affect the performance in the benchmarks. We present a detailed overview of differences in the experimental setup of the related work in Table 4 in the appendix.

To better understand the effect of sampling algorithms, we thus carry out experiments on a fixed GCN architecture (Kipf & Welling, 2016) with two layers, a hidden size of 256, batch size of 256, sample size of 256 nodes per layer, and a fixed number of epochs per dataset (detailed in Appendix B). Under this setting, we compare GRAPES with the following baselines: a Random baseline that uses the same setup as GRAPES but with uniform inclusion probabilities; FastGCN (Chen et al., 2018b), LADIES (Zou et al., 2019), GraphSAINT (Zeng et al., 2019), GAS (Fey et al., 2021), AS-GCN (Huang et al., 2018), and PASS Yoon et al. (2021). With this setup, we aim to control our experiments in a way such that variations in performance can only be attributed to the sampling method.

For all baselines (except Random), we rely on their publicly available implementations. For all methods, we optimize the learning rate using the performance on the validation set. GRAPES requires selecting two additional hyperparameters: the learning rate for the sampling policy and, for GRAPES-GFN, the reward scaling parameter α. We tune these based on the classification performance on the validation set. These hyperparameters are specific to our sampling method and do not explicitly affect the learning capacity of the GCN classifier. We refer the reader to Appendix B for more details on hyperparameter settings.

6.1 Datasets

Most methods in the literature of sampling in GNNs are evaluated on *homophilous* graphs (where the connected nodes are likely to have similar labels) and multi-class classification, where nodes are classified into one of several classes (in contrast to multi-label). To further understand the effect of sampling, we carry out experiments with *heterophilous* graphs, where nodes that are connected differ in their features and labels,

Table 1: F1-scores (%) for different sampling methods trained on **homophilous** graphs, for a batch size of 256, and sample size of 256 per layer. We report the mean and standard deviation over 10 runs. The best values among the sampling baselines (all except GAS) are in **bold**, and the second best are underlined. MC stands for multi-class and ML stands for multi-label classification. OOM indicates out-of-memory.

Dataset Homophily Task	**Cora** $h = 0.81$ MC	**Citeseer** $h = 0.74$ MC	**Pubmed** $h = 0.80$ MC	**Reddit** $h = 0.78$ MC	**ogbn-arxiv** $h = 0.65$ MC	**ogbn-products** $h = 0.81$ MC	**DBLP** $h = 0.76$ ML
GAS	87.00 ± 0.19	85.87 ± 0.19	87.45 ± 0.23	94.75 ± 0.04	68.36 ± 0.55	74.69 ± 0.14	83.08 ± 0.31
FastGCN	76.17 ± 3.98	62.81 ± 7.19	53.52 ± 28.48	62.93 ± 3.28	39.49 ± 8.04	66.09 ± 3.04	62.93 ± 3.28
LADIES	76.02 ± 11.69	63.48 ± 12.21	72.81 ± 17.67	59.30 ± 2.69	43.52 ± 8.03	68.08 ± 1.95	59.97 ± 10.45
GraphSAINT	87.28 ± 0.49	77.28 ± 0.67	87.45 ± 0.75	91.47 ± 0.94	63.54 ± 1.75	67.66 ± 0.69	77.09 ± 0.56
AS-GCN	85.60 ± 0.54	**79.21 ± 0.19**	**90.58 ± 0.40**	93.52 ± 0.40	**65.38 ± 1.80**	**73.94 ± 0.40**	**83.24 ± 0.51**
PASS	82.03 ± 0.074	77.17 ± 0.69	88.33 ± 0.45	OOM	58.32 ± 0.05	OOM	54.40 ± 2.31
Random	86.58 ± 0.33	78.29 ± 0.52	90.09 ± 0.17	94.16 ± 0.06	61.35 ± 0.32	70.47 ± 0.32	76.87 ± 0.24
GRAPES-RL (ours)	**87.62 ± 0.48**	78.75 ± 0.35	89.40 ± 0.34	94.09 ± 0.05	62.58 ± 0.64	71.45 ± 0.20	76.88 ± 0.33
GRAPES-GFN (ours)	87.29 ± 0.32	78.75 ± 0.33	89.46 ± 0.34	**94.30 ± 0.06**	61.86 ± 0.51	70.66 ± 0.30	77.14 ± 0.48
Rank	1	2	3	1	3	2	2

Table 2: F1-scores (%) for different sampling methods trained on **heterophilous** graphs for a batch size of 256, and a sample size of 256 per layer. We report the mean and standard deviation over 10 runs. The best values among the sampling baselines (all except GAS) are in **bold**, and the second best are underlined. MC stands for multi-class and ML stands for multi-label classification. OOM indicates out of memory.

Dataset Homophily Task	**Flickr** $h = 0.31$ MC	**snap-patents** $h = 0.22$ MC	**Yelp** $h = 0.22$ ML	**ogbn-proteins** $h = 0.15$ ML	**BlogCat** $h = 0.1$ ML
GAS	49.96 ± 0.28	38.04 ± 0.20	37.81 ± 0.07	7.55 ± 0.01	6.07 ± 0.04
FastGCN	46.40 ± 2.51	29.68 ± 0.61	29.29 ± 4.39	7.80 ± 0.55	6.44 ± 0.41
LADIES	47.19 ± 3.42	29.09 ± 1.88	18.92 ± 3.18	4.31 ± 0.11	6.74 ± 0.67
GraphSAINT	48.01 ± 1.44	28.01 ± 0.57	34.68 ± 0.70	9.94 ± 0.07	6.89 ± 0.96
AS-GCN	48.42 ± 1.20	**31.04 ± 0.19**	38.51 ± 1.45	5.20 ± 0.36	5.43 ± 0.54
PASS	44.49 ± 0.76	OOM	OOM	7.71 ± 0.12	6.88 ± 0.59
Random	49.39 ± 0.23	29.74 ± 0.34	40.63 ± 0.13	10.82 ± 0.05	7.13 ± 0.97
GRAPES-RL (ours)	**49.54 ± 0.67**	29.16 ± 0.54	40.69 ± 0.55	**11.78 ± 0.14**	9.06 ± 0.68
GRAPES-GFN (ours)	49.29 ± 0.32	29.58 ± 0.24	**44.57 ± 0.88**	11.57 ± 0.18	**9.22 ± 0.40**
Rank	1	4	1	1	1

and multi-label datasets in which a node can be assigned one or more labels. As shown in the literature (Zhu et al., 2020; Zhao et al., 2023), we argue that node classification with GCN is more challenging on these datasets compared to multi-class classification on homophilous graphs, where an adaptive policy could be beneficial. In particular, we run experiments with the following datasets for the node classification task:

- **Homophilous** graphs: citation networks (Cora, Citeseer, Pubmed with the "full" split) (Sen et al., 2008; Yang et al., 2016), Reddit (Hamilton et al., 2017), ogbn-arxiv and ogbn-products (Hu et al., 2020), and DBLP (Zhao et al., 2023).
- **Heterophilous** graphs: Flickr (Zeng et al., 2019), Yelp (Zeng et al., 2019), ogbn-proteins (Hu et al., 2020), BlogCat (Zhao et al., 2023), and snap-patents (Leskovec & Krevl, 2014).

We included the *edge homophily* ratio (Zhu et al., 2020) of all the datasets in Table 1 and 2. For the multi-label datasets, we followed the same framework in (Zhao et al., 2023) to calculate their label homophily ratio. Statistics about the datasets can be found in Appendix C.

6.2 Results

Comparison with sampling methods We present the F1 scores of GCNs trained via sampling for GRAPES and the sampling-based baselines in Table 1 and 2. We observe that on the majority of heterophilous datasets, and all the heterophilous multi-label datasets, both variations of GRAPES achieve the highest F1-score. As we mentioned earlier, node classification on heterophilous graphs is a challenging task for GCNs, due to the diversity of the neighbor nodes' labels. Our results suggest that GRAPES' ability to adapt the sampling policy to particular features of the data allows GRAPES to learn the complex patterns across the heterophilous datasets. Table 1 shows that on most homophilous graphs, AS-GCN achieves the highest F1-score, indicating the importance of adaptive sampling. However, unlike GRAPES, this method is only adaptive to the sampling variance and fails to outperform GRAPES on the heterophilous graphs. Moreover, our results show that PASS, another adaptive method, is unable to compete with GRAPES and results in an out-of-memory error on several large datasets. This is due to its node-wise sampling feature, which results in a disconnected sampled graph.

FastGCN and LADIES, which use a fixed policy, fail to compete with the other methods in our experimental setup. They assign a higher probability to nodes with higher degrees. This heuristic results in neglecting informative low-degree nodes. Although outperforming LADIES and FastGCN, GraphSAINT, which also uses a fixed sampling policy, fails to outperform the more competitive baselines. We observe that on some datasets, Random sampling achieves a comparable F1-score to GRAPES and the baselines. This mainly happens on simple datasets, such as homophilous graphs, indicating that on these graphs, a simple sampling approach is sufficient.

Comparison with a non-sampling scalable method GAS (Fey et al., 2021) is a non-sampling method that uses the historical embeddings of the 1-hop neighbors of the target nodes. We present classification results in Table 2, and we visualize memory usage in Fig. 3. While GAS has higher classification F1-scores for some datasets, GRAPES achieves comparable F1-scores and significantly outperforms GAS for Yelp, ogbn-proteins, and BlogCat, all heterophilous multi-label graphs. Once again, these results indicate the effectiveness of the adaptivity of GRAPES in sampling influential nodes in complex graphs. For the memory usage, we compare GAS with AS-GCN and the GFlowNet version of GRAPES-32 and GRAPES-256, where 32 and 256 (the default value) indicate the sample size per layer. The memory usage results show that, even with a large sample size, GRAPES can use up to an order of magnitude less GPU memory than GAS, especially for large datasets. In large, densely connected graphs such as Reddit, the 1-hop neighborhood can be massive. Then, the difference in memory use for GRAPES, which only sees a small set of neighbors, and GAS, which uses all the neighbors, is significant. While GAS occasionally achieves higher F1-scores, it consistently demands more memory, indicating a potential compromise between accuracy and computational efficiency. In contrast, GRAPES strikes a balance, delivering comparable F1-score with more modest memory footprints. In most cases, GRAPES also uses less memory than AS-GCN, another sampling-based baseline. In other cases, GRAPES uses more memory than AS-GCN, which we attribute to the fact that there is an additional GNN in GRAPES for learning the sampling policy. Some variations are due to implementation details[4], though in general we observe a significant advantage of layer-wise sampling methods over historical embeddings.

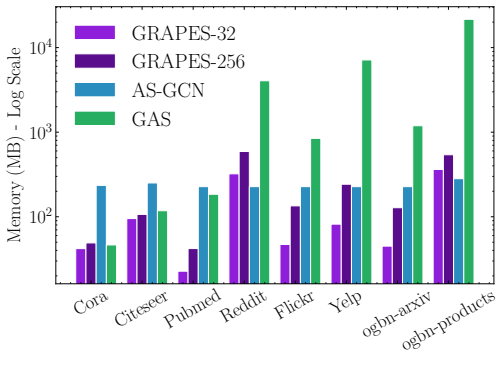

Figure 3: GPU peak memory allocation (MB) for GAS, and GRAPES-GFN-32 and GRAPES-GFN-256.

GRAPES is robust to low sample sizes. A desirable property of sampling methods, in contrast with full-batch GNNs or methods like GAS (which relies on historical embeddings), is the ability to control the sample size to reduce memory usage as needed. To study this property, we show in Figure 4 the effects of

[4]We used PyTorch for our implementation, whereas AS-GCN is implemented in Tensorflow, which manages GPU memory differently.

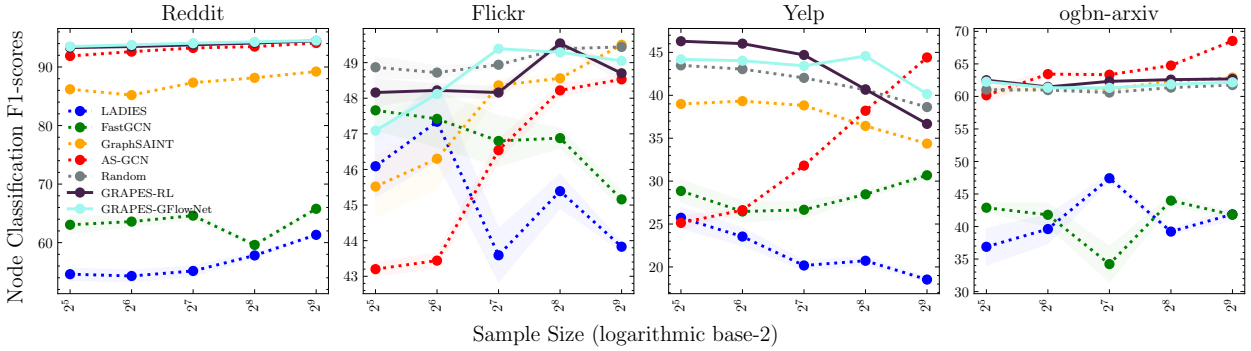

Figure 4: Comparative analysis of classification accuracy across different sampling sizes for sampling baseline and GRAPES. We repeated each experiment five times: The shaded regions show the 95% confidence intervals.

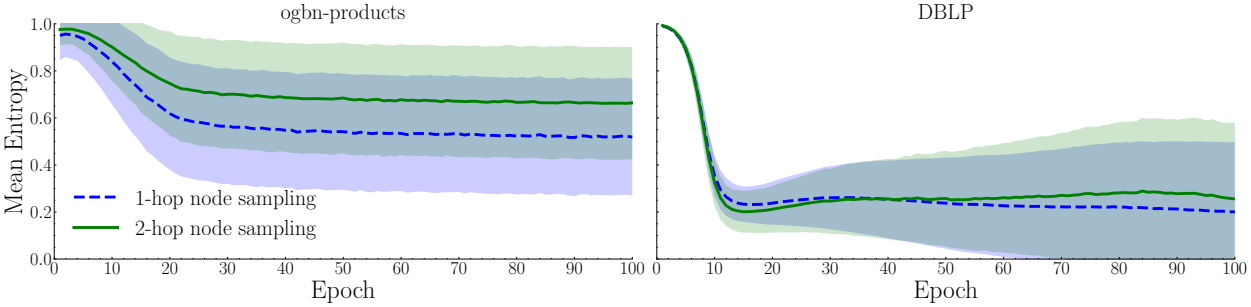

Figure 5: Entropy for the ogbn-products and DBLP datasets. The mean is the entropy in bits of the node probability, averaged over nodes. The shaded region indicates the standard deviation of the entropy over all nodes.

varying the sample size on Reddit, Flickr, Yelp, and ogbn-arxiv. Our results show that both the RL and GFN variants of GRAPES are robust to low sample size, and achieve strong performance with fewer sampled nodes needed than the baselines, enabling training GCNs on larger graphs while using less GPU memory. Random sampling also exhibits robustness to sample size for all datasets except Yelp, where accuracy drops in larger sample sizes. GRAPES-RL shows the same behavior while performing slightly better than Random on Yelp. AS-GCN and GraphSAINT show the largest dependence on sample size, especially on Flickr and Yelp.

GRAPES learns strong preferences over nodes. The ability to selectively choose *influential* nodes is a crucial property of GRAPES. Figure 5 shows the mean and standard deviation of base 2 entropy for the node preference probabilities for the two layers of GCN_S for ogbn-products and DBLP for GRAPES-GFN. The probabilities show preference towards particular nodes with a Bernoulli distribution. A well-trained model must have a high preference (probability close to 1) for some nodes and a low preference (probability close to 0) for the rest. Therefore, we would like a low average entropy with a high standard deviation. As the figure shows, the mean entropy in both layers decreases from almost 1 and converges to a value above 0, while the standard deviation increases. This indicates that for ogbn-products, the sampling policy initially assigned a probability near 0.5, indicating little preference. However, after several training epochs, GRAPES starts preferring some nodes, resulting in lower mean entropy. We observe similar behavior for most datasets (DBLP, BlogCat, Yelp, ogbn-arxiv, and ogbn-proteins). However, we observe that the sampling policy exhibits no preferences among the nodes for the other datasets. For more details about the other datasets, see Appendix D.

7 Known Limitations and Future Work

Our experimental results show that GRAPES outperforms the baselines on some, but not all, heterophilous datasets. We provide a theoretical analysis that shows the effectiveness of adaptive sampling on certain types of heterophilous graphs. Further analysis is required to understand which graph characteristics contribute to the performance gain observed with adaptive sampling methods such as GRAPES.

Moreover, in our experiments, we focused only on the problem of node classification. However, GRAPES is not tied to a particular downstream task. GRAPES assumes access to a tractable reward function (Bengio et al., 2021b). Therefore, GRAPES will be applicable for other graph-related tasks, like link prediction and unsupervised representation learning.

Additionally, there is room for a more thorough investigation of the properties of the subgraphs that GRAPES samples. The experiments at the end of Section 6.2 and in Appendix D are a first step towards this. We leave these directions for future work.

Another direction for future work is applying GRAPES to architectures designed to address heterophily, such as the ones proposed by Zhu et al. (2020); Abu-El-Haija et al. (2019). By leveraging GRAPES, these architectures could potentially overcome their scalability issues.

8 Discussion, Broader Impacts, and Conclusion

We propose GRAPES, an adaptive graph sampling method based on reinforcement learning and GFlowNet, facilitating the scalability of training GNNs on massive graphs. GRAPES samples a subgraph of influential nodes by learning node preferences that adapt to classification loss, which depends on node features, GNN architecture, classification task, and graph topology. Our experiments demonstrate that GRAPES effectively selects nodes from large-scale graphs and achieves state-of-the-art performance in multi-label classification tasks on heterophilous graphs, while performing competitively for multi-class classification on homophilous graphs. Compared to the other sampling methods, GRAPES can maintain high classification accuracy even with lower sample sizes, indicating GRAPES' ability to scale to larger graphs by sampling a small but influential set of nodes. GRAPES achieves comparable performance to GAS, while using up to an order of magnitude less memory.

Focus on Heterophily in Sampling. Previous works have overlooked the impact of sampling on heterophilous and multi-label graphs. To our knowledge, this is the first work to compare sampling effects on both homophilous and heterophilous graphs, where adaptive sampling is especially crucial due to the diversity among neighbors, making node selection vital for GNN accuracy.

Lack of Uniform Evaluation Protocol. Existing methods in the literature report performance on graph sampling under settings with different GCN architectures, regularization techniques, feature normalization strategies, and data splits, among other differences. These differences made it challenging to determine the benefits of each sampling method. This motivated us to implement a unified protocol across all methods, where we keep the architecture fixed. We encourage future work to consider a similar methodology for a fair evaluation, or an experimental review study, as is common in other areas of machine learning research on graphs (Shchur et al., 2018; Ruffinelli et al., 2019).

Broader Impacts. Our paper focuses on fundamental research to understand and advance sampling in large-scale graphs. There may be multiple potential societal consequences of our work, but there are none that can be easily predicted and specifically highlighted.

Acknowledgements

Taraneh Younesian was funded by Huawei DREAMS Lab. All content represents the opinion of the authors, which is not necessarily shared nor endorsed by their respective employers and/or sponsors in Huawei DREAMS Lab. Daniel Daza was partially funded by Elsevier's Discovery Lab. Emile van Krieken was

funded by ELIAI (The Edinburgh Laboratory for Integrated Artificial Intelligence), EPSRC (grant no. EP/W002876/1). We thank Michael Cochez and Ruud van Bakel for insightful discussions.

References

Sami Abu-El-Haija, Bryan Perozzi, Amol Kapoor, Nazanin Alipourfard, Kristina Lerman, Hrayr Harutyunyan, Greg Ver Steeg, and Aram Galstyan. Mixhop: Higher-order graph convolutional architectures via sparsified neighborhood mixing. In *international conference on machine learning*, pp. 21–29. PMLR, 2019.

Sami Abu-El-Haija, Joshua V Dillon, Bahare Fatemi, Kyriakos Axiotis, Neslihan Bulut, Johannes Gasteiger, Bryan Perozzi, and Mohammadhossein Bateni. Submix: Learning to mix graph sampling heuristics. In *Uncertainty in Artificial Intelligence*, pp. 1–10. PMLR, 2023.

Kareem Ahmed, Zhe Zeng, Mathias Niepert, and Guy Van den Broeck. SIMPLE: A gradient estimator for k-subset sampling. In *The Eleventh International Conference on Learning Representations*, 2023.

Aamir Anis, Akshay Gadde, and Antonio Ortega. Efficient sampling set selection for bandlimited graph signals using graph spectral proxies. *IEEE Transactions on Signal Processing*, 64(14):3775–3789, 2016.

Emmanuel Bengio, Moksh Jain, Maksym Korablyov, Doina Precup, and Yoshua Bengio. Flow network based generative models for non-iterative diverse candidate generation. *Advances in Neural Information Processing Systems*, 34:27381–27394, 2021a.

Yoshua Bengio, Nicholas Léonard, and Aaron C. Courville. Estimating or propagating gradients through stochastic neurons for conditional computation. *CoRR*, abs/1308.3432, 2013. URL http://arxiv.org/abs/1308.3432.

Yoshua Bengio, Salem Lahlou, Tristan Deleu, Edward J Hu, Mo Tiwari, and Emmanuel Bengio. Gflownet foundations. *arXiv preprint arXiv:2111.09266*, 2021b.

Lukas Biewald. Experiment tracking with weights and biases, 2020. URL https://www.wandb.com/. Software available from wandb.com.

Jianfei Chen, Jun Zhu, and Le Song. Stochastic training of graph convolutional networks with variance reduction. In *International Conference on Machine Learning*, pp. 942–950. PMLR, 2018a.

Jie Chen, Tengfei Ma, and Cao Xiao. FastGCN: Fast learning with graph convolutional networks via importance sampling. In *6th International Conference on Learning Representations, ICLR 2018, Conference Track Proceedings*. OpenReview.net, 2018b.

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 *Proceedings of the 25th ACM SIGKDD international conference on knowledge discovery & data mining*, pp. 257–266, 2019.

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

Zicun Cong, Baoxu Shi, Shan Li, Jaewon Yang, Qi He, and Jian Pei. Fairsample: Training fair and accurate graph convolutional neural networks efficiently. *IEEE Transactions on Knowledge and Data Engineering*, 36(4):1537–1551, 2023.

Tristan Deleu, António Góis, Chris Emezue, Mansi Rankawat, Simon Lacoste-Julien, Stefan Bauer, and Yoshua Bengio. Bayesian structure learning with generative flow networks. In *Uncertainty in Artificial Intelligence*, pp. 518–528. PMLR, 2022.

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

Matthias Fey, Jan E Lenssen, Frank Weichert, and Jure Leskovec. Gnnautoscale: Scalable and expressive graph neural networks via historical embeddings. In *International conference on machine learning*, pp. 3294–3304. PMLR, 2021.

Wenhao Gao, Tianfan Fu, Jimeng Sun, and Connor Coley. Sample efficiency matters: a benchmark for practical molecular optimization. *Advances in Neural Information Processing Systems*, 35:21342–21357, 2022.

Haoyu Geng, Chao Chen, Yixuan He, Gang Zeng, Zhaobing Han, Hua Chai, and Junchi Yan. Pyramid graph neural network: A graph sampling and filtering approach for multi-scale disentangled representations. In *Proceedings of the 29th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pp. 518–530, 2023.

Will Hamilton, Zhitao Ying, and Jure Leskovec. Inductive representation learning on large graphs. *Advances in Neural Information Processing Systems*, 30, 2017.

Kurt Hornik, Maxwell B. Stinchcombe, and Halbert White. Multilayer feedforward networks are universal approximators. *Neural Networks*, 2(5):359–366, 1989. doi: 10.1016/0893-6080(89)90020-8. URL https://doi.org/10.1016/0893-6080(89)90020-8.

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. *Advances in neural information processing systems*, 33:22118–22133, 2020.

Wenbing Huang, Tong Zhang, Yu Rong, and Junzhou Huang. Adaptive sampling towards fast graph representation learning. *Advances in neural information processing systems*, 31, 2018.

Iris AM Huijben, Wouter Kool, Max B Paulus, and Ruud JG Van Sloun. A review of the gumbel-max trick and its extensions for discrete stochasticity in machine learning. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 45(2):1353–1371, 2022.

Moksh Jain, Emmanuel Bengio, Alex Hernandez-Garcia, Jarrid Rector-Brooks, Bonaventure FP Dossou, Chanakya Ajit Ekbote, Jie Fu, Tianyu Zhang, Michael Kilgour, Dinghuai Zhang, et al. Biological sequence design with gflownets. In *International Conference on Machine Learning*, pp. 9786–9801. PMLR, 2022.

Moksh Jain, Tristan Deleu, Jason Hartford, Cheng-Hao Liu, Alex Hernandez-Garcia, and Yoshua Bengio. Gflownets for AI-driven scientific discovery. *Digital Discovery*, 2(3):557–577, 2023.

Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

Thomas N Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*, 2016.

Jure Leskovec and Andrej Krevl. SNAP Datasets: Stanford large network dataset collection. http://snap.stanford.edu/data, 2014.

Michelle M Li, Kexin Huang, and Marinka Zitnik. Graph representation learning in biomedicine and healthcare. *Nature Biomedical Engineering*, 6(12):1353–1369, 2022.

Wenqian Li, Yinchuan Li, Zhigang Li, Jianye Hao, and Yan Pang. Dag matters! gflownets enhanced explainer for graph neural networks. *arXiv preprint arXiv:2303.02448*, 2023.

Ziqi Liu, Zhengwei Wu, Zhiqiang Zhang, Jun Zhou, Shuang Yang, Le Song, and Yuan Qi. Bandit samplers for training graph neural networks. *Advances in Neural Information Processing Systems*, 33:6878–6888, 2020.

Zirui Liu, Kaixiong Zhou, Zhimeng Jiang, Li Li, Rui Chen, Soo-Hyun Choi, and Xia Hu. Dspar: An embarrassingly simple strategy for efficient gnn training and inference via degree-based sparsification. *Transactions on Machine Learning Research*, 2023.

Lu Ma, Zeang Sheng, Xunkai Li, Xinyi Gao, Zhezheng Hao, Ling Yang, Xiaonan Nie, Jiawei Jiang, Wentao Zhang, and Bin Cui. Acceleration algorithms in gnns: A survey. *IEEE Transactions on Knowledge and Data Engineering*, 2025.

Nikolay Malkin, Moksh Jain, Emmanuel Bengio, Chen Sun, and Yoshua Bengio. Trajectory balance: Improved credit assignment in gflownets. *Advances in Neural Information Processing Systems*, 35:5955–5967, 2022a.

Nikolay Malkin, Salem Lahlou, Tristan Deleu, Xu Ji, Edward J Hu, Katie E Everett, Dinghuai Zhang, and Yoshua Bengio. Gflownets and variational inference. In *The Eleventh International Conference on Learning Representations*, 2022b.

Shakir Mohamed, Mihaela Rosca, Michael Figurnov, and Andriy Mnih. Monte carlo gradient estimation in machine learning. *Journal of Machine Learning Research*, 21:132:1–132:62, 2020.

David F Nettleton. Data mining of social networks represented as graphs. *Computer Science Review*, 7:1–34, 2013.

Daniel Ruffinelli, Samuel Broscheit, and Rainer Gemulla. You can teach an old dog new tricks! on training knowledge graph embeddings. In *International Conference on Learning Representations*, 2019.

Luana Ruiz, Luiz FO Chamon, and Alejandro Ribeiro. Transferability properties of graph neural networks. *IEEE Transactions on Signal Processing*, 2023.

Prithviraj Sen, Galileo Namata, Mustafa Bilgic, Lise Getoor, Brian Galligher, and Tina Eliassi-Rad. Collective classification in network data. *AI magazine*, 29(3):93–93, 2008.

Marco Serafini and Hui Guan. Scalable graph neural network training: The case for sampling. *ACM SIGOPS Operating Systems Review*, 55(1):68–76, 2021.

Oleksandr Shchur, Maximilian Mumme, Aleksandar Bojchevski, and Stephan Günnemann. Pitfalls of graph neural network evaluation. *arXiv preprint arXiv:1811.05868*, 2018.

Zhihao Shi, Xize Liang, and Jie Wang. Lmc: Fast training of gnns via subgraph sampling with provable convergence. *arXiv preprint arXiv:2302.00924*, 2023.

Petar Velickovic, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, Yoshua Bengio, et al. Graph attention networks. *stat*, 1050(20):10–48550, 2017.

Tim Vieira. Gumbel-max trick and weighted reservoir sampling, 2014. URL http://timvieira.github.io/blog/post/2014/08/01/gumbel-max-trick-and-weighted-reservoir-sampling/.

Yu Wang, Zhiwei Liu, Ziwei Fan, Lichao Sun, and Philip S Yu. Dskreg: Differentiable sampling on knowledge graph for recommendation with relational gnn. In *Proceedings of the 30th ACM International Conference on Information & Knowledge Management*, pp. 3513–3517, 2021.

Ronald J. Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning*, 1992. ISSN 0885-6125. doi: 10.1007/bf00992696.

Shiwen Wu, Fei Sun, Wentao Zhang, Xu Xie, and Bin Cui. Graph neural networks in recommender systems: a survey. *ACM Computing Surveys*, 55(5):1–37, 2022.

Zhilin Yang, William Cohen, and Ruslan Salakhudinov. Revisiting semi-supervised learning with graph embeddings. In *International conference on machine learning*, pp. 40–48. PMLR, 2016.

Minji Yoon, Théophile Gervet, Baoxu Shi, Sufeng Niu, Qi He, and Jaewon Yang. Performance-adaptive sampling strategy towards fast and accurate graph neural networks. In *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*, pp. 2046–2056, 2021.

Haoran You, Zhihan Lu, Zijian Zhou, Yonggan Fu, and Yingyan Lin. Early-bird gcns: Graph-network co-optimization towards more efficient gcn training and inference via drawing early-bird lottery tickets. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 36(8), pp. 8910–8918, 2022.

Haiyang Yu, Limei Wang, Bokun Wang, Meng Liu, Tianbao Yang, and Shuiwang Ji. Graphfm: Improving large-scale gnn training via feature momentum. In *International Conference on Machine Learning*, pp. 25684–25701. PMLR, 2022.

Seongjun Yun, Minbyul Jeong, Raehyun Kim, Jaewoo Kang, and Hyunwoo J Kim. Graph transformer networks. *Advances in neural information processing systems*, 32, 2019.

Hanqing Zeng, Hongkuan Zhou, Ajitesh Srivastava, Rajgopal Kannan, and Viktor Prasanna. GraphSAINT: Graph sampling based inductive learning method. *arXiv preprint arXiv:1907.04931*, 2019.

Guibin Zhang, Kun Wang, Wei Huang, Yanwei Yue, Yang Wang, Roger Zimmermann, Aojun Zhou, Dawei Cheng, Jin Zeng, and Yuxuan Liang. Graph lottery ticket automated. In *The Twelfth International Conference on Learning Representations*, 2024.

Tianqi Zhao, Ngan Thi Dong, Alan Hanjalic, and Megha Khosla. Multi-label node classification on graph-structured data. *Trans. Mach. Learn. Res.*, 2023, 2023.

Xin Zheng, Yi Wang, Yixin Liu, Ming Li, Miao Zhang, Di Jin, Philip S Yu, and Shirui Pan. Graph neural networks for graphs with heterophily: A survey. *arXiv preprint arXiv:2202.07082*, 2022.

Jiong Zhu, Yujun Yan, Lingxiao Zhao, Mark Heimann, Leman Akoglu, and Danai Koutra. Beyond homophily in graph neural networks: Current limitations and effective designs. *Advances in neural information processing systems*, 33:7793–7804, 2020.

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 *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019*, pp. 11247–11256, 2019.

A Off-Policy Sampling Setup

In this Appendix, we discuss the technical and mathematical challenges around our setup that resulted in our off-policy learning setup. In each layer l of the GFlowNet, we aim to sample exactly k out of n nodes. An initially natural setup would be to use the distribution over k-subsets of $\mathcal{N}(K^{(l-1)})$ (Ahmed et al., 2023). Using Bayes theorem,

$$q(\mathcal{V}^{(l)}|\mathcal{V}^{(0)},\ldots,\mathcal{V}^{(l-1)},k) = \frac{I[|\mathcal{V}^{(l)}|=k]q(\mathcal{V}^{(l)}|\mathcal{V}^{(0)},\ldots,\mathcal{V}^{(l-1)})}{\sum_{\mathcal{V}'^{(l+1)}} I[|\mathcal{V}'^{(l+1)}|=k]q(\mathcal{V}^{(l)}|\mathcal{V}^{(0)},\ldots,\mathcal{V}^{(l-1)})}. \tag{7}$$

When conditioned on k, q assigns 0 probability to sets of nodes $\mathcal{V}^{(l)}$ that do not sample exactly k new nodes (that is, when $|\mathcal{V}^{(l)}| \neq k$). However, this requires renormalizing the distribution, which is the function of the denominator term on the right-hand side. Note that this sum is over an exponential number of elements, namely $2^{|\mathcal{N}(K^{(l-1)})|}$, and naive computation is clearly intractable. SIMPLE (Ahmed et al., 2023) provides an optimized dynamic programming algorithm for computing this normalization constant. However, it scales polynomially in $|\mathcal{N}(K^{(l-1)})|$ and k, and in our experiments, computing the normalizer is a bottleneck already for mid-sized graphs like Reddit.

Therefore, we decided to circumvent having to compute $q(\mathcal{V}^{(l)}|\mathcal{V}^{(0)},\ldots,\mathcal{V}^{(l-1)},k)$ by sampling using the Gumbel-Top-k trick (Equation 4) to ensure we always add exactly k nodes. However, we are now in an off-policy setting: The samples using Equation 4 are distributed by $q(\mathcal{V}^{(l)}|\mathcal{V}^{(0)},\ldots,\mathcal{V}^{(l-1)},k)$, not by $q(\mathcal{V}^{(l)}|\mathcal{V}^{(0)},\ldots,\mathcal{V}^{(l-1)})$, and so we sample from a different distribution than the one we use to compute the loss. Previous work (Malkin et al., 2022b) showed that the Trajectory Balance loss is amenable to off-policy training without importance sampling and weighting without introducing high variance. This is important since importance weighting would require us to weight by $q(\mathcal{V}^{(l)}|\mathcal{V}^{(0)},\ldots,\mathcal{V}^{(l-1)})/q(\mathcal{V}^{(l)}|\mathcal{V}^{(0)},\ldots,\mathcal{V}^{(l-1)},k)$, reintroducing the need to compute $q(\mathcal{V}^{(l)}|\mathcal{V}^{(0)},\ldots,\mathcal{V}^{(l-1)},k)$.

The off-policy benefits of the Trajectory Balance loss provide a strong argument over more common Reinforcement Learning setups. Off-policy training in Reinforcement Learning usually requires importance weighting to be stable, which is not tractable in our setting.

B Experimental Details

For all experiments, we used as architecture the Graph Convolutional Network (Kipf & Welling, 2016), with two layers, a hidden size of 256, a batch size of 256, and a sampling size of 256 nodes per layer. We implemented the GCNs in GRAPES via PyTorch Geometric (Fey & Lenssen, 2019). We train for 50 epochs on Cora, Citeseer, and Reddit; 100 epochs on BlogCat, DBLP, Flickr, ogbn-products, Pubmed, snap-patents, and Yelp; and 150 epochs on ogbn-arxiv and ogbn-proteins. The ogbn-proteins and BlogCat datasets do not contain node features, and instead we learn node embeddings for them of dimension 128 for ogbn-proteins, and 64 for BlogCat.

Our experiments were carried out in a single-node cluster setup. We conducted our experiments on a machine with Nvidia RTX A4000 GPU (16GB GPU memory), Nvidia A100 (40GB GPU memory), and Nvidia RTX A6000 GPU (48GB GPU memory) and each machine had 48 CPUs. In total, we estimate that our experiments took 200 compute days.

B.1 Hyperparameter Tuning

We tune the hyperparameters of GRAPES using a random search strategy with the goal of maximizing the accuracy of the validation dataset. We used Weights and Biases for hyperparameter tuning [5]. The best-performing hyperparameters for every dataset can be found in our repository https://anonymous.4open.science/r/GRAPES. The following are the hyperparameters that we tuned: the learning rate of the GFlowNet, the learning rate of the classification GCN, and the scaling parameter α. We used the log uniform distribution to sample the aforementioned hyperparameters with the values from the following ranges, respectively, $[1e-6, 1e-2]$, $[1e-6, 1e-2]$, and [1e2, 1e6]. We kept the other hyperparameters, such as the batch size and hidden dimension of the GCN. We used the Adam optimizer (Kingma & Ba, 2014) for GCN_C and GCN_S.

We did a hyperparameter sensitivity analysis performed using Weights & Biases Biewald (2020) for training of GRAPES-GFN on Yelp. Importance measures how strongly each parameter influences the validation accuracy (higher = more critical), while correlation shows the direction and magnitude of the linear relationship (positive = increasing the parameter tends to increase the accuracy, negative = the opposite). From the table below, lr_{gc}, the learning rate for the classification GCN, has the strongest effect on accuracy and is negatively correlated, suggesting that higher learning rates can harm performance. Meanwhile, lr_{gf} and α show smaller impacts and weaker correlations, indicating that the model is less affected by changes in these two parameters.

Table 3: Importance and correlation of configuration parameters for training GRAPES-GFN on Yelp.

Config Parameter	Importance	Correlation
lr_gc	0.633	-0.610
lr_gf GCN)	0.280	0.055
α	0.087	0.097

B.2 Baselines

For a fair comparison, we adjusted the implementations of the baselines so that the only difference is the sampling methods and the rest of the training conditions are kept the same. In the following, we explain the details of the modifications to each of the baselines.

For LADIES, we used the official implementation, which also contains an implementation of FastGCN. We changed the nonlinear activation function from ELU to ReLU, and we removed any linear layers after the two layers of the GCN, set dropout to zero, and disabled early stopping. We also noticed that the original LADIES implementation divided the target nodes into mini-batches, not from the entire graphs as we do,

[5] https://wandb.ai

Table 4: Differences in experimental setups in related work, obtained from the original publications and their official implementations. *This indicates the total budget in terms of nodes sampled across all layers.

	FastGCN (Chen et al., 2018b)	**LADIES** (Zou et al., 2019)	**GraphSAINT** (Zeng et al., 2019)	**AS-GCN** (Huang et al., 2018)	**GAS** (Fey et al., 2021)
Architecture	GCN	GCN	GCN	GCN+attention	GCN,GCNII,GAT,GIN,APPNP,PNA
Hidden size	(16, 128)	256	(128, 256, 512, 2048)	(16,256)	(256, 512, 1024, 2048)
Number of layers	2	5	(2, 4, 5)	2	(2, 4, 64)
Batch size	(256, 1024)	512	(400, 512, 1000)	256	(1,2,5,40,12,100)
Nodes per layer	(100, 400)	(5, 64, 512)	(4500, 6000, 8000)*	(128, 256, 512)	No sampling
Training epochs	(100, 200, 300)	300	2000	(50,100,300)	(300, 400, 500, 1000)

but into random fragments. This means that LADIES and FastGCN do not see all the target nodes in the training data. We kept this setting unchanged because otherwise it would have significantly slowed down the training of these two methods.

For GraphSAINT, we noticed that the GNN consists of two layers of higher-order aggregators, which are a combination of GraphSage-mean (Hamilton et al., 2017) and MixHop (Abu-El-Haija et al., 2019), and a linear classification layer at the end. Moreover, the original implementation of GraphSAINT is only applicable to inductive learning on the graphs, where the training graph only contains the training nodes and is entirely different from the validation and test graphs, where only the nodes from the validation set and test are available, respectively. We argue that in transductive learning, unlike inductive learning, the motivation to scale to larger graphs is higher since the validation and test nodes are also available during training, and therefore, the processed graph is larger. Finally, to keep all the configurations the same as GRAPES, we used PyTorch Geometric's function for GraphSAINT Random Walk sampler with depth two, which showed the best performance across the three variations of GraphSAINT. Therefore, we use the same GCN and data loader (transductive) as ours and use GraphSAINT to sample a subgraph of nodes for training. We use the node sampler setting since it is the only setting that allows specifying different sampling budgets, and therefore, can be compared to layer-wise methods with the same sampling budget. We also removed the early stopping and used the same number of epochs as GRAPES. Note that for Figure 4, we need to change the sample size while the Random Walk sampler can only sample equal to the batch size per layer. Therefore, for these experiments, we used the Node Sampler version of GraphSAINT, which allows us to change the sample size. For instance, for sample size 32 per layer, we set GraphSAINT's Node Sampler's sample size to $256 + 32 + 32 = 320$ to sample a subgraph with 320 nodes. Please refer to our repository for more details about the implementation of GraphSAINT.

For GAS, we used the original implementation. However, we changed the configuration of the GCN to have a two-layer GCN with 256 hidden units. We turned off dropout, batch normalization and residual connections in the GCN. We also removed early stopping for the training.

For AS-GCN we removed the attention mechanism used in the GCN classifier. Their method also uses attention in the sampler, which is separate from the classifier, so we keep it.

For PASS, we changed the inference to full-batch to match the rest of the baselines. We set the parameter `sampling_scope` to 512, because in the original experiments of the paper, this value was set to twice the batch size. For Flickr, ogbn-arxiv, and ogbn-proteins, this value results in OOM; therefore, we set it to 128, 128, and 256, respectively. For the datasets that we report OOM in tables 1 and 2, we tested all possible `sampling_scope` between 512 and 16.

C Dataset statistics

We present the statistics of the datasets used in our experiments in Table 5. The splits that we used for Cora, Citeseer, and Pubmed correspond to the "full" splits, in which the label rate is higher than in the "public" splits. For BlogCat, we take the average accuracy of all the methods across the three available splits provided by (Zhao et al., 2023). For DBLP and snap-patents, we use the average of ten random splits because these two datasets had no predefined splits.

Table 5: Statistics of the datasets used in our experiments. The label rate indicates the percentage of nodes used for training. ogbn-proteins and BlogCat do not contain node features, and instead, we learn embeddings for nodes in these datasets.

Dataset	Task	Nodes	Edges	Features	Classes	Label Rate (%)
Cora	multi-class	2,708	5,278	1,433	7	44.61
CiteSeer	multi-class	3,327	4,552	3,703	6	54.91
PubMed	multi-class	19,717	44,324	500	3	92.39
Reddit	multi-class	232,965	11,606,919	602	41	65.86
DBLP	multi-label	28,702	136,670	300	4	80.00
Flickr	multi-class	89,250	449,878	500	7	50.00
snap-patents	multi-class	2,923,922	27,945,092	269	5	50.00
Yelp	multi-label	716,847	6,977,409	300	100	75.00
ogbn-proteins	multi-label	132,534	79,122,504	—	112	65.30
BlogCat	multi-label	10,312	667,966	—	39	60.00
ogbn-arxiv	multi-class	169,343	1,157,799	128	40	53.70
ogbn-products	multi-class	2,449,029	61,859,076	100	47	8.03

D Entropy as Node preference measure

Figures 6 and 7 show the mean and standard deviation of entropy in base two of all the datasets. We calculate the mean entropy as the following:

$$E = \frac{1}{n}\sum_{i=1}^{n} p_i \cdot \log_2(p_i) + (1-p_i)\cdot \log_2(1-p_i) \tag{8}$$

where n is the number of neighbors of the nodes sampled in the previous layer and p_i is the probability of inclusion for each node, which is the output of the GFlowNet. As the figures show, for certain small datasets (Cora, Citeseer, Pubmed, Flickr) the mean entropy is: 1) very close to 1, indicating that GRAPES prefers every nodes with the probability close to 0.5, or 2) close to 0 but also with a low standard deviation, meaning that it equally prefers the majority of the nodes with the probability 1 or 0. On the contrary, for the large datasets (Reddit, Yelp, ogbn-arxiv, ogbn-products) by the end of training, the average entropy is lower than 1, with a standard deviation around 0.3 indicating that GRAPES learns different preferences over different nodes, some with a probability close to 1, and some close to 0.

We analyzed the label distribution of the sampled subgraphs via GRAPES-GF for Yelp, by counting the percentage of nodes having each label and calculating the difference between the original and sampled graphs, i.e., percentage of nodes having label c in the original graph minus percentage of nodes having label c in the sampled graph. As shown in the figure, all the values are positive, indicating that a higher ratio of nodes have each label in the original graph. Moreover, this difference increases at the end of training. This means that among the 100 labels in Yelp, GRAPES prefers the nodes with fewer labels or that are even single-labeled. This may make the classification task easier and result in a lower loss. Moreover, among the 100 classes, some are sampled the most and some the least. Figure 8 below shows this trend.

E GPU Memory usage comparison between GRAPES and GAS

We compared different variants of GRAPES, with different sample sizes (32, 256), with GAS (Fey et al., 2021), which is a non-sampling method. Figure 3 shows the GPU memory allocation (MB), on a logarithmic scale for GRAPES-32, GRAPES-256, and GAS. The three graph methods exhibit distinct performance characteristics across various datasets. We used the `max_memory_allocated` function in PyTorch to measure the GPU

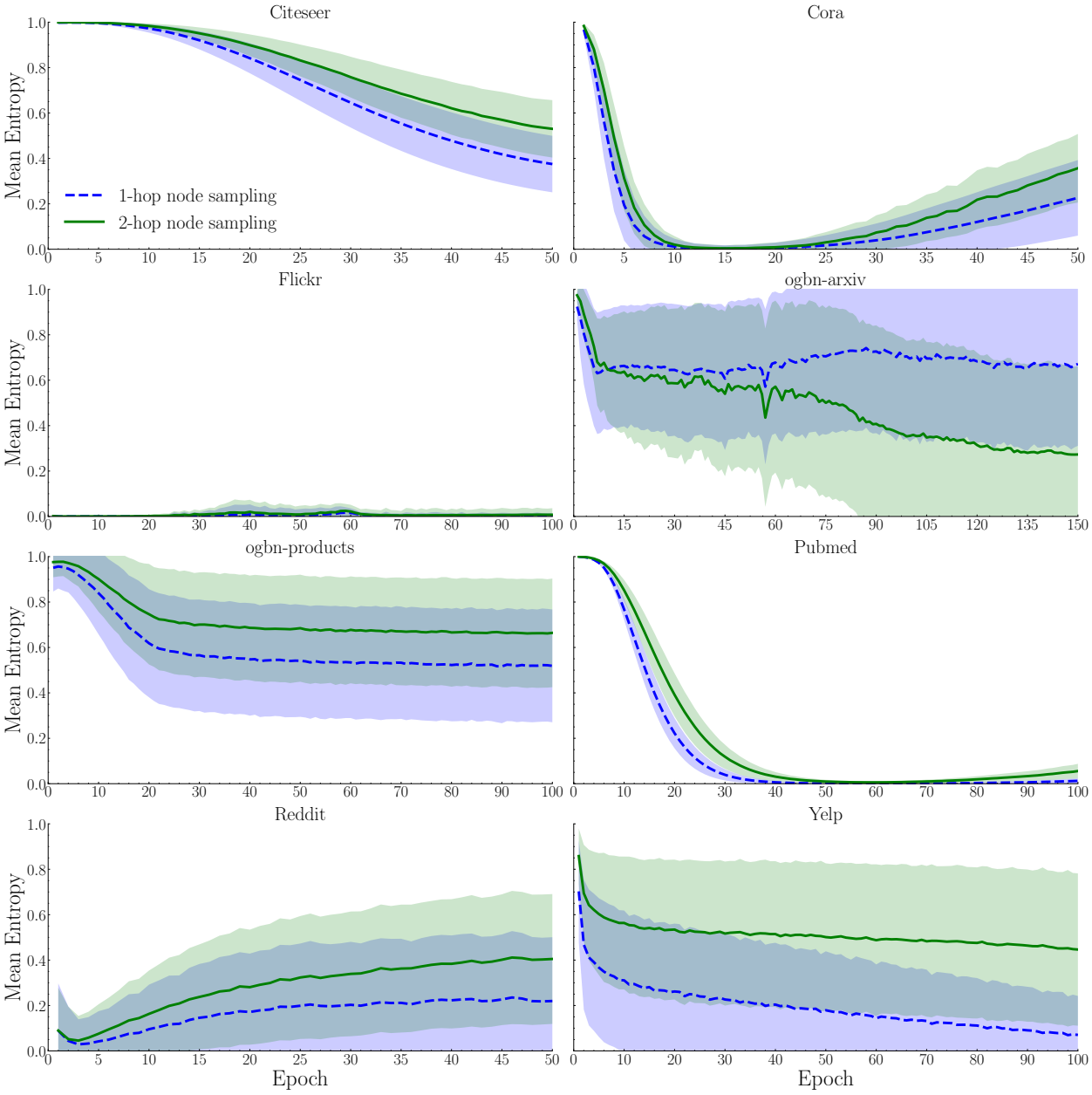

Figure 6: Combined entropy plots for Citeseer, Cora, Flickr, ogbn-arxiv, ogbn-products, Pubmed, Reddit, and Yelp showcasing the mean entropy. The shaded region indicates the standard deviation of the entropy *across nodes*. The plots compare 1-hop node sampling against 2-hop node sampling.

memory allocation.[6] Since this function measures the maximum memory allocation since the beginning of the program, where the memory measurement is done is not relevant.

F GFlowNet details

This section provides additional details on GFlowNets in general and the GFlowNet version of GRAPES.

[6] https://pytorch.org/docs/stable/generated/torch.cuda.max_memory_allocated.html

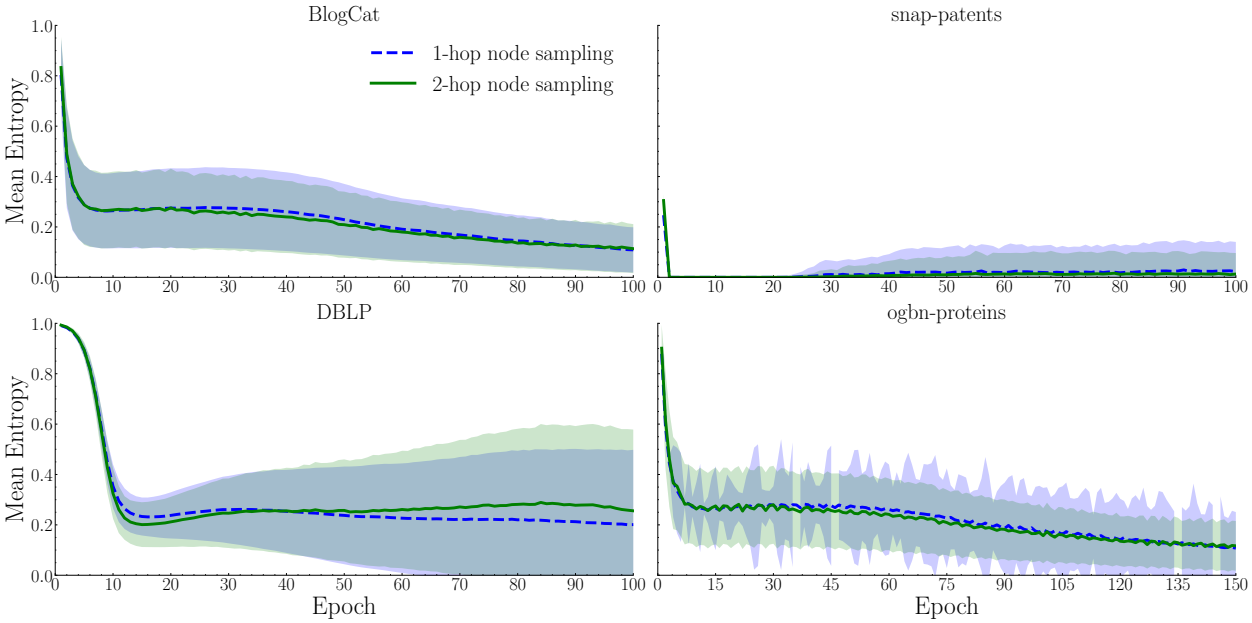

Figure 7: Combined entropy plots for BlogCat, snap-patents, DBLP, and ogbn-proteins, showcasing the mean entropy across epochs. The shaded region indicates the standard deviation of the entropy *across nodes*. The plots compare 1-hop node sampling against 2-hop node sampling.

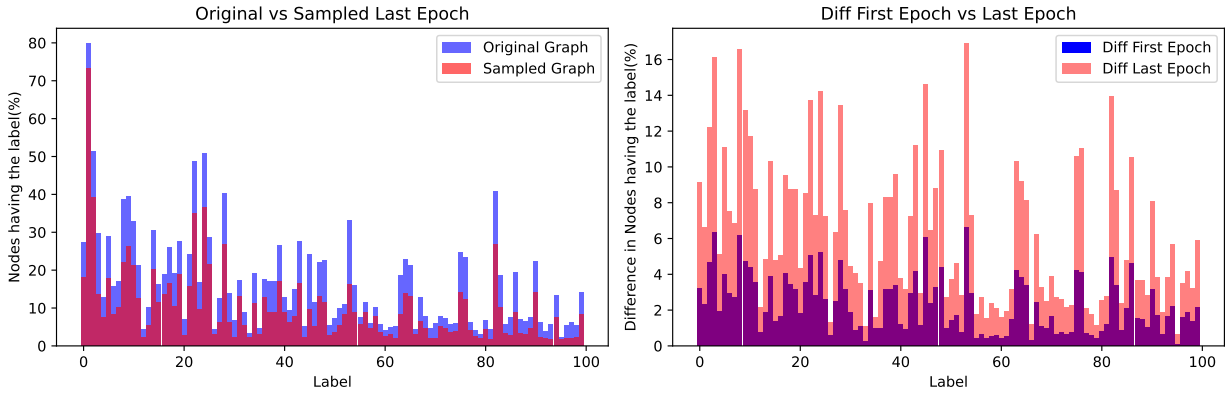

Figure 8: Comparison of Label distributions between GRSAPES-GFN and the original graph for Yelp. The left figure shows the percentage of nodes in the last epoch of both graphs that have each label. The right shows the difference between those percentages in the original graph and the sampled graph at the beginning and end of training.

F.1 Generative Flow Networks

Generative Flow Networks (GFlowNets) (Bengio et al., 2021a;b) are generative models that can sample from a very large structured space. GFlowNets construct the structure in multiple generation steps. Compared to Reinforcement Learning approaches, GFlowNets learn to sample *in proportion* to a given reward function, while in Reinforcement Learning, reward functions are maximized. This feature of GFlowNets encourages sampling diverse sets of high-quality structures, instead of only considering the single best structure. GFlowNets have been used in several applications like molecule design and material science (Bengio et al., 2021a; Gao et al., 2022; Jain et al., 2022), Bayesian structure learning (Deleu et al., 2022), scientific discovery (Jain et al., 2023), and GNN explainability (Li et al., 2023). Similar to our work, the latter utilizes a GFlowNet to sample

subgraphs. However, this method *explains* a trained GNN and is not used to scale GNN training to large graphs.

F.2 GFlowNet and Trajectory Balance Loss

We first give a brief overview of GFlowNets (Bengio et al., 2021a;b) and the trajectory balance loss (Malkin et al., 2022a). Let $\mathcal{G}_F = (\mathcal{S}, \mathcal{A}, \mathcal{S}_0, \mathcal{S}_f, R)$ denote a GFlowNet learning problem. Here, $\mathcal{S}$ is a finite set of states that forms a directed graph with $\mathcal{A}$, a set of directed edges representing actions or transitions between states. $\mathcal{S}_0 \subset \mathcal{S}$ is the set of initial states, $\mathcal{S}_f \subset \mathcal{S}$ is the set of terminating states,[7] and $R : \mathcal{S}_f \to \mathbb{R}_+$ is the reward function defined on terminating states. At time t, a particular $a_t \in \mathcal{A}$ indicates the action taken to transition from state $s^{(t-1)}$ to $s^{(t)}$. A trajectory τ is a path through the graph from an initial state $s^{(0)}$ to a terminating state $s^{(n)} \in \mathcal{S}_f$: $\tau = (s^{(0)} \to \ldots \to s^{(n)})$. A GFlowNet is a neural network that learns to transition from an initial state $s^{(0)}$ to a terminating state where the reward $R(s^{(n)})$ is given. The goal of the GFlowNet is to ensure that following the forward transition probabilities $P_F(s^{(t)}|s^{(t-1)})$ leads to final states $s \in \mathcal{S}_f$ with probability in proportion to the reward R (Bengio et al., 2021a). The *Trajectory Balance (TB)* loss (Malkin et al., 2022a) is developed with this goal. For a trajectory $\tau = (s^{(0)} \to \ldots \to s^{(n)})$, the TB loss is:

$$\mathcal{L}_{TB}(\tau) = \left(\log \frac{Z(s^{(0)}) \prod_{t=1}^{n} P_F(s^{(t)}|s^{(t-1)})}{R(s^{(n)}) \prod_{t=1}^{n} P_B(s^{(t-1)}|s^{(t)})}\right)^2, \tag{9}$$

where $Z : \mathcal{S}_0 \to \mathbb{R}_+$ computes the total flow of the network from the starting state $s^{(0)}$ and P_F and P_B are the forward and the backward transition probabilities between the states, where both can be parameterized by a neural network (Malkin et al., 2022a).

F.3 GFlowNet Design: States, Actions, and Reward

Next, we explain our choice of $\mathcal{G}_F$, that is, the states, actions, terminating states, and reward function, and the coupling of our GFlowNet with the sampling policy q. A state $s^{(l-1)} \in \mathcal{S}$ represents a sequence of sets of nodes $s^{(l-1)} = (\mathcal{V}^{(0)}, \ldots, \mathcal{V}^{(l-1)})$ sampled so far. An action from $s^{(l-1)}$ represents choosing k nodes without replacement among $\mathcal{N}(K^{(l-1)})$. This forms the set of nodes $\mathcal{V}^{(l)}$ in the next layer. Therefore, in an L layer GCN, we construct a sequence of L sets of nodes to reach a terminating state.

We define the optimal sampling policy as having the lowest classification loss in expectation. Therefore, a set of k nodes with a lower classification loss than another set must have a higher probability. Our goal is to design $\mathcal{G}_F$ so that it learns the forward and backward transition probabilities proportional to a given reward. We define the reward function as below:

$$R(s^{(L)}) = R(\mathcal{V}^{(0)}, \ldots, \mathcal{V}^{(L)}) := \exp(-\alpha \cdot \mathcal{L}_{\mathrm{C}}(X, Y, K^{(0)}, \ldots, K^{(L)})), \tag{10}$$

where $\mathcal{L}_{\mathrm{C}}$ is the classification loss and α is a *scaling parameter*, which we explain in Section F.4.

Forward Probability. The forward probability $P_F(s^{(l-1)}|s^{(l)})$ in GRAPES is the sampling policy $q(\mathcal{V}^{(l)}|\mathcal{V}^{(0)}, \ldots, \mathcal{V}^{(l-1)})$ defined in Section 4.1.

Backward Probability. Trajectory balance (Equation 9) also requires defining the probability of transitioning backwards through the states. The backward probability is a distribution over all parents of a state. This distribution is not required in our setup, as the state representation $s^{(l)} = (\mathcal{V}^{(0)}, \ldots, \mathcal{V}^{(l)})$ saves the trajectory taken through $\mathcal{G}_F$ to get to $s^{(l)}$. This means the graph for the GFlowNet learning problem $\mathcal{G}_F$ is a tree, as each state $s^{(l)} = (\mathcal{V}^{(0)}, \ldots, \mathcal{V}^{(l-1)}, \mathcal{V}^{(l)})$ has exactly 1 parent, namely $s^{(l-1)} = (\mathcal{V}^{(0)}, \ldots, \mathcal{V}^{(l-1)})$. Since each state has a single parent, we find that $P_B(s^{(l-1)}|s^{(l)}) = 1$ when we retrace the trajectory. We pass the information on when a node is added to the GFlowNet by adding an identifier to the nodes' embeddings that indicates in what layer it was sampled, as explained in Section 4.1.

[7] Technically, GFlowNets have unique source and terminal (or 'sink') states s_s and s_f. The source state has an edge to all initial states, and all terminating states have an edge to the terminal state.

Table 6: Comparison between GRAPES and variants using the straight-through estimator (STE).

	Cora	Flickr	ogbn-arxiv	Yelp
Best GRAPES	87.62 ± 0.48	49.54 ± 0.67	62.58 ± 0.64	44.57 ± 0.88
STE	87.95 ± 0.18	46.33 ± 1.39	61.95 ± 0.32	37.29 ± 0.15
STE_N	87.03 ± 0.35	46.29 ± 1.47	60.13 ± 0.41	15.63 ± 0.10

Loss derivation Combining our setup with the trajectory balance loss (Equation 9), the GRAPES loss is

$$\mathcal{L}_{\text{GFN}}(X, Y, \mathcal{V}^{(0)}) = \left(\log \frac{Z(\mathcal{V}^{(0)}) \prod_{l=1}^{L} P_F(s^{(l)}|s^{(l-1)})}{R(s^{(l)})} \right)^2 \tag{11}$$

$$= \Big(\log Z(\mathcal{V}^{(0)}) + \sum_{l=1}^{L} \log q(\mathcal{V}^{(l)}|\mathcal{V}^{(0)}, \ldots, \mathcal{V}^{(l-1)}) + \tag{12}$$

$$\alpha \cdot \mathcal{L}_{\text{C}}(X, Y, K^{(0)}, \ldots, K^{(L)}) \Big)^2$$

We model the initial-state-dependent normalizer $Z(s^{(0)})$ in Equation 9 with a trainable GCN, namely $\text{GCN}_Z(\mathcal{V}^{(0)})$. It predicts the normalizer conditioned on the target nodes given. It is trained together with GCN_S by minimizing the trajectory balance loss.

We note that, like in GRAPES-RL, we use off-policy sampling from $q(\mathcal{V}^{(l)}|\mathcal{V}^{(0)}, \ldots, \mathcal{V}^{(l-1)}, k)$ to train the GFlowNet. See Appendix A for additional details.

F.4 Reward scaling

In our experiments, we noticed that with the bigger datasets, the GFlowNet is more affected by the log-probabilities than the reward from the classification GCN_C. The reason that the term $\log q(\mathcal{V}^{(l)}|\mathcal{V}^{(0)}, \ldots, \mathcal{V}^{(l-1)})$ dominates $\mathcal{L}_{\text{GFN}}$ is that $\log q(\mathcal{V}^{(l)}|\mathcal{V}^{(0)}, \ldots, \mathcal{V}^{(l-1)}) = \sum_{i \in \mathcal{N}(K^{(l-1)})} \log p_i$, which sums over $|\mathcal{N}(K^{(l-1)})|$ elements. Given a batch size of 256, this neighborhood can be as big as $52,000$ nodes, resulting in summing $52,000$ log-probabilities. The majority of the probabilities p_i are values close to zero. Therefore, the above sum would be a large negative number. Since the loss $\mathcal{L}_\text{C}$ is, in our experiments, often quite close to zero, the log-probability and its variance dominate the loss. Therefore, we add the hyperparameter α to the reward and tune it in our experiments.

G GRAPES and the straight-through estimator

The straight-through estimator Bengio et al. (2013) (STE) has been proposed as a method for learning in computation graphs involving a non-differentiable function f of an input x, by replacing f with the identity in the backward pass during computation of the gradients.

In practice, applying the STE for learning in GRAPES requires ensuring that the gradients flow through the sampled nodes, for example by treating the top-k sampled nodes at each layer as a mask which multiplies the messages from these nodes to *all* their neighbors. Such a computation over all neighboring nodes defeats the purpose of sampling, increasing memory usage.

An alternative to keep memory usage low is to keep the gradients only for the top-k probabilities, though this results in a biased estimate of the gradient. We implemented two versions of this approach. In the first version (STE), we simply use the perturbed log probability (log probability plus the Gumbel noise) of the top-k nodes as additional weights for the classification GCN. In the second version, we normalized these weights to have a mean equal to one (STE_N). We present results in Table 6. While in Cora the results are relatively similar to our implementation of GRAPES, as we experiment with larger graphs, there is a significant drop in performance when employing either variant of the STE.

H Proof of Theorem 1

H.1 Proof

In this section, we prove Theorem 1 in section 5 as follows:

Proof. Let $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ be an undirected fully connected simple graph with an even number of nodes N, $\mathcal{V} = \{1, \ldots, N\}$, a set of edges $\mathcal{E} = \{e_{ij}\}_{i,j=1}^{N}$, where e_{ij} denotes an edge between v_i and v_j. Let $Y = \{y_i | i \in \{1, \ldots, N\}, y_i \in \{0, 1\}\}$ be the set of node labels and let $X \in \mathbb{R}^{N \times 2}$ be the node features. Let $\mathcal{E}_1$ and $\mathcal{E}_2$ be one partition of $\mathcal{E}$, where $\mathcal{E}_1$ consists of $N/2$ edges with two conditions: 1) $e_{ij} \in \mathcal{E}_1$, if and only if the first feature of nodes v_i and v_j are equal, i.e. $e_{ij} \in \mathcal{E}_1 \Leftrightarrow x_{i1} = x_{j1}$, and 2) each node appears only at one edge in $\mathcal{E}_1$. The second condition implies that each value for the first feature appears only in the nodes connected by one edge in $\mathcal{E}_1$, i.e., for two distinct edges $e_{ij}, e_{kl} \in \mathcal{E}_1$, $x_{1i} \neq x_{1k}$.

For each edge e_{ij} in $\mathcal{E}_1$, sample $x_{i2} \sim \text{Bern}(0.5)$ on $\{0, 1\}$. Then, set the node label y_j to x_{i2}. That is, the label of each node is equal to the second feature of its corresponding neighbor in $\mathcal{E}_1$. Since $\mathcal{G}$ is fully connected, the L-hop neighborhood for all nodes is $\mathcal{V}$. Therefore, non-adaptive methods cannot prefer any node over another: a GCN with a non-adaptive sampler can only achieve accuracy above chance by sampling K nodes such that for each target node, the neighbor with the same first feature is sampled. The probability of sampling such a neighbor is $p = \frac{\binom{N-2}{K-1}}{\binom{N-1}{K}} = \frac{K}{N-1}$. The nominator indicates choosing the desired neighbor connected to the target node in $\mathcal{E}_1$ and then choosing the remaining $K-1$ nodes among the $N-2$ nodes. The denominator indicates choosing K nodes among $N-1$ nodes (excluding the target node). Repeating the sampling for L layers, eliminating K nodes from the candidate neighbors for sampling for the next layer, will result in $p = \sum_{i=0}^{L-1} \frac{K}{N-iK-1} \leq \frac{LK}{N}$.

However, there is a GCN with an adaptive sampler that classifies nodes with an accuracy of 100%. Specifically, such an adaptive sampler compares the first node feature of the neighbors with the target node v_i, and then always samples the unique v_j where $e_{ij} \in \mathcal{E}_1$. Subsequently, it samples $K-1$ additional neighbors randomly. If the sampler perfectly samples the informative neighbors in $\mathcal{E}_1$, we assume there exists a GCN that can classify the target nodes correctly. We show a construction of such a GCN in section H.2. Figure 9 shows an example of such a graph with eight nodes and a homophily ratio of 0.43. □

In the above proof, we show that there are graphs for which there is a performance gap between adaptive and non-adaptive samplers. Usually $LK \ll N$; therefore, the probability $p \leq \frac{LK}{N}$ for non-adaptive sampling methods to perform better than the chance level becomes very small. These graphs are heterophilous, with an expected homophily ratio equal to 0.5. We calculate this expectation below:

$$\mathbb{E}(h_{\mathcal{G}}) = \mathbb{E}\left(\frac{\binom{m}{2} + \binom{N-m}{2}}{\binom{N}{2}}\right), m \sim \text{Binomial}(N, 0.5)$$

For that, we need $\mathbb{E}\left(\binom{m}{2}\right) = \frac{1}{2}\mathbb{E}\left(m(m-1)\right)$. Because $\mathbb{E}(m) = \frac{N}{2}$ and $Var(m) = \frac{N}{4}$, we have

$$\mathbb{E}(m(m-1)) = \mathbb{E}(m^2 - m) = \mathbb{E}(m^2) - \mathbb{E}(m) = Var(m) + (\mathbb{E}(m))^2 - \mathbb{E}(m)$$

$$= \frac{N}{4} + \left(\frac{N}{2}\right)^2 - \frac{N}{2} = \frac{N^2 - N}{4},$$

so we have

$$\mathbb{E}\left(\binom{m}{2}\right) = \frac{1}{2} \times \frac{N^2 - N}{4} = \frac{N^2 - N}{8}.$$

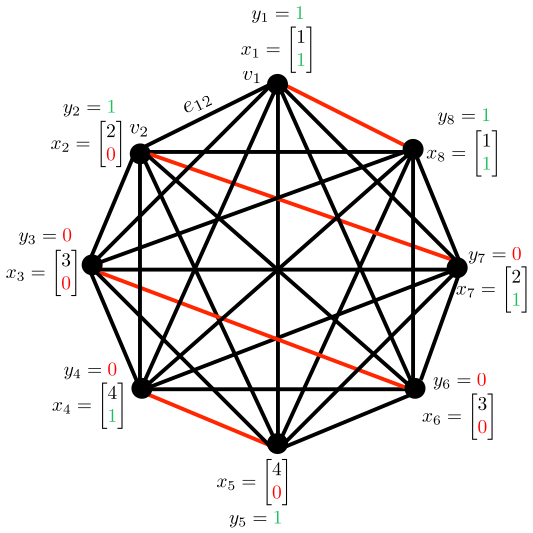

Figure 9: An example of a graph for Theorem 1 with eight nodes. Red edges belong to $\mathcal{E}_1$, features x_i and labels y_i are shown beside every node. For nodes v_1 and v_2 we show the edge e_{12} as an example. As shown, the label of each node is the second feature of its neighbor, where a red edge connects them. The edge homophily ratio is $h = \frac{12}{28} = 0.43$.

Similarly, $\mathbb{E}\left(\binom{N-m}{2}\right) = \frac{N^2-N}{8}$. Therefore, the expected homophily ratio is

$$\mathbb{E}(h_{\mathcal{G}}) = \frac{\frac{N^2-N}{8} + \frac{N^2-N}{8}}{\binom{N}{2}} = 0.5.$$

H.2 GCN Construction

In this section, we construct a GCN that can compute the classification rule assumed in the proof of Theorem 1. Let $\mathbf{x}_i \in \mathbb{R}^2$ be an input feature vector of a node $v_i \in \mathcal{V}$. We construct a GCN where the first layer computes a 4-dimensional vector for a target node v_i:

$$\mathbf{h}_i^{(1)} = \tilde{\mathbf{W}}_1 \mathbf{x}_i + \frac{1}{|\mathcal{N}(v_i)|} \sum_{v_j \in \mathcal{N}(v_i)} \mathbf{W}_1 \mathbf{x}_j. \tag{13}$$

Here, $\tilde{\mathbf{W}}_1$ is the self-loop weight, and $\mathbf{W}_1$ is the weight for the neighbors of a target node. We define the following:

$$\tilde{\mathbf{W}}_1 = \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ 0 & 0 \\ 0 & 0 \end{bmatrix}, \quad \mathbf{W}_1 = \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 1 & 0 \\ 0 & 1 \end{bmatrix} \tag{14}$$

For a given target node, the output of this layer is a vector $\mathbf{h}_i^{(1)} \in \mathbb{R}^4$ where the first two values correspond to the target node's features, and the other two values are the average of the features of its neighbors. The next layer implements the following function: if values $h_{i1}^{(1)}$ and $h_{i3}^{(1)}$ are the same, then the output is the second value of this vector, otherwise it is zero. We denote this function as $h_i^{(2)} = f(\mathbf{h}_i^{(1)}) \in \mathbb{R}$, which can be implemented with a multi-layer perceptron due to its universal approximation properties (Hornik et al., 1989). An MLP can be implemented in a GCN by omitting the neighbors and using a self-loop weight matrix only. The last layer of the GCN selects the average value in the neighborhood of a target node from the scalars computed by f, which can be computed using a simple GCN where the self-loop weight is set to zero: $h_i^{(3)} = \frac{1}{|\mathcal{N}(v_i)|} \sum_{v_j \in \mathcal{N}(v_i)} h_j^{(2)}$. This output corresponds to the classification rule required for Theorem 1.