SubMix: Learning to Mix Graph Sampling Heuristics

Sami Abu-El-Haija[1] **Joshua V. Dillon**[1] **Bahare Fatemi**[2] **Kyriakos Axiotis**[3] **Neslihan Bulut**[1]
Johannes Gasteiger[4] **Bryan Perozzi**[2] **Mohammadhossein Bateni**[2]

[1]Google Research, Mountain View, California, USA
[2]Google Research, Montreal, Quebéc, Canada
[3]Google Research, New York, NY, USA
[4]Google Research, Zürich, Switzerland

Abstract

Sampling subgraphs for training Graph Neural Networks (GNNs) is receiving much attention from the GNN community. While a variety of methods have been proposed, each method samples the graph according to its own heuristic. However, there has been little work in mixing these heuristics in an end-to-end trainable manner. In this work, we design a generative framework for subgraph sampling. Our method, SubMix, parameterizes subgraph sampling as a convex combination of heuristics. We show that a continuous approximation of the discrete sampling process allows us to efficiently obtain analytical gradients for training the sampling parameters. Our experimental results illustrate the usefulness of learning graph sampling in three scenarios: (1) robust training of GNNs by automatically learning to discard noisy edge sources; (2) improving model performance by trainable and online edge subset selection; and (3) by integrating our framework into decoupled GNN models improves their performance on standard benchmarks.

1 INTRODUCTION

Graph Neural Networks (GNNs) are frequently used as machine learning models for relational data, *e.g.*, in predicting node classes or relationships between nodes (*a.k.a.* edges). The node and edge types are domain-specific. Application domains include social, biochemical, and computational networks, *e.g.*, where GNNs are used for recommendation, predicting protein-protein interactions, or job scheduling [Bruna et al., 2014, Kipf and Welling, 2017, Monti et al., 2017, Veličković et al., 2018, Hamilton et al., 2017, Battaglia et al., 2018, Chami et al., 2022].

Early GNNs were conceptually formulated and implemented on the graph as a whole [Bruna et al., 2014, Kipf and Welling, 2017]. As such, the entire graph is processed at every training step. For training on larger graphs, researchers resorted to a variety of directions, including (D.1) **subgraph sampling** and (D.2) **graph-decoupled training**, among others (§7).

Direction (D.1) repeatedly samples *subgraphs* from the (larger) input graph. Each subgraph sample serves as one training example for the GNN. The process of sampling subgraphs–how to select a meaningful neighborhood for a set of nodes–is usually based on a heuristic and is not trainable. For instance, Chiang et al. [2019] partitions the input graph into smaller subgraphs via an algorithm for graph clustering such as METIS [Karypis and Kumar, 1998]. Once the subgraphs are created, they are repeatedly used throughout training (and inference). Zeng et al. [2020] try three samplers: node-level, edge-level, and random walk-based samplers, respectively, similar to [Chen et al., 2018], [Zou et al., 2019] and [Hamilton et al., 2017, Markowitz et al., 2021]. These methods all employ sampling heuristics, *e.g.*, choose nodes with probability proportional to their degree [Chen et al., 2018], or a few edges for every sampled node [Hamilton et al., 2017, Ying et al., 2018, Markowitz et al., 2021]. While these heuristics stem from reasonable inductive biases, learning to combine them is under-explored.

Contributions[1]: We propose a generative model of subgraphs (§3), which can be written as a mixture distribution of sampling heuristics, where the mixture weights are learned. Given a node, each heuristic assigns a probability distribution on its neighbors. We train parameters of the mixture model to optimize the supervision objective, *e.g.*, cross-entropy for node classification. To obtain analytical gradients, we relax the discrete process of subgraph sampling into the continuous domain. We evaluate our method in three setups. First, if we use an adversary sampling heuris-

[1]Our implementation could be accessed through `https://github.com/tensorflow/gnn/tree/main/tensorflow_gnn/models/submix`

Proceedings of the 39th Conference on Uncertainty in Artificial Intelligence (UAI 2023), PMLR 216:1–10.

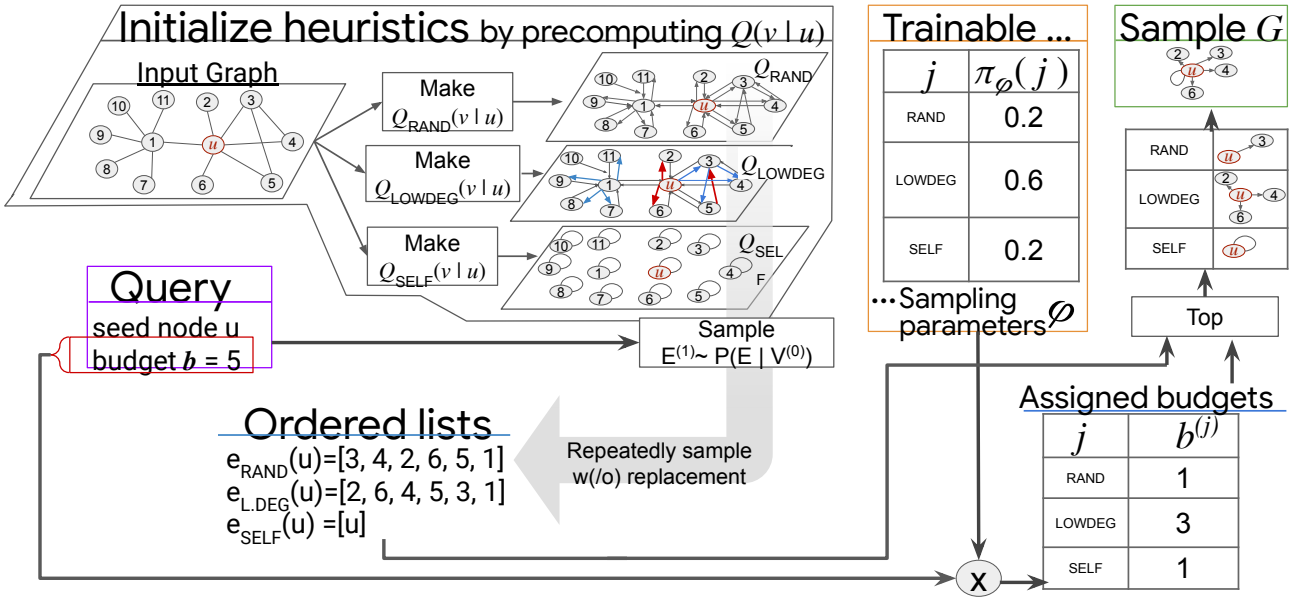

Figure 1: **Summary of our Method**. **(top-left)**. The input graph initializes our utilized heuristics only once. Each heuristic maintains distribution $Q(\cdot \mid u)$ per node u. We show relative mass outgoing a node by edge color: red and blue, for Q_{LOWDEG}, depict larger mass than grey edges. Once initialized, **(left)**, user can request 5 edges outgoing from node u. Each heuristic j then yields an **ordered list** of neighbors, sampled from $Q_j(\cdot \mid u)$ with (or without) replacement. The **(Trainable)** $\pi_\phi(j)$ allocates the total budget of 5 among the heuristics where heuristic j can select $b_j = \pi_\phi(j) \times 5$. **(top right)** Finally, the sampled edges get combined into sampled subgraph. The discrete process is relaxed in §3.4.

tic, always yielding non-edges, we find that our method assigns it weight ≈ 0. Second, we cluster edges using features of its connecting endpoint nodes. Our method can choose which cluster to sample more frequently, generally increasing model performance. Finally, we show how our framework can be integrated into full-graph decoupled GNN methods (D.2) This improves the performance of decoupled GNNs on standard benchmarks. The performance gains introduce no additional GNN parameters.

2 BACKGROUND

Denote the space of graphs of n nodes as $\mathcal{G} = \mathcal{A} \times \mathcal{X}$. A graph $\mathbf{G} \in \mathcal{G}$ with adjacency matrix $\mathbf{A} \in \mathcal{A} \stackrel{\text{def}}{=} \mathbb{R}^{n \times n}$ node feature matrix $\mathbf{X} \in \mathcal{X} \stackrel{\text{def}}{=} \mathbb{R}^{n \times d}$ with d-features per node. Denote node set as $\mathcal{V} \stackrel{\text{def}}{=} [n]$. Node $u \in \mathcal{V}$ has an edge to node $v \in \mathcal{V}$ iff $A_{uv} = 1$. For node classification task with c classes, let $\mathcal{Y} \stackrel{\text{def}}{=} \mathbb{R}^{n \times c}$ define the label space. Denote labeled examples as $\mathcal{V}_{\text{tr}} \subseteq \mathcal{V}$. Let $\mathcal{N}(u)$ denote the set of out-neighbors of u with $\mathcal{N}(u) = \{A_{u,v} = 1 \mid v \in \mathcal{V}\}$.

2.1 GRAPH NEURAL NETWORKS

GNNs are neural networks for graph data structures, where nodes repeatedly communicate their features [Chami et al., 2022] to their neighbors, where each node incorporates all its incoming messages into its representation. GNNs, for node classification tasks, can be trained by minimizing the negative log-likehood:

$$-\log p_\theta(\mathbf{Y} \mid \mathbf{A}, \mathbf{X}). \tag{1}$$

A popular GNN architecture is the GCN of [Kipf and Welling, 2017]:

$$p_\theta(\mathbf{Y} \mid \mathbf{A}, \mathbf{X}) = \text{softmax}(\widehat{\mathbf{A}}[\widehat{\mathbf{A}}\mathbf{X}\mathbf{W}^{(1)}]_+\mathbf{W}^{(2)}), \tag{2}$$

where $\theta = \{\mathbf{W}^{(1)}, \mathbf{W}^{(2)}\}$, $\widehat{\cdot}$ is symmetric normalization[2] $\widehat{\mathbf{A}} = \mathbf{D}^{-\frac{1}{2}}\mathbf{A}\mathbf{D}^{-\frac{1}{2}}$ and $\mathbf{D} = \text{diag}(\mathbf{1}^\top\mathbf{A})$. Another popular GNN model of GraphSAGE [Hamilton et al., 2017]:

$$p_\theta(\mathbf{Y} \mid \mathbf{A}, \mathbf{X}) = \text{softmax}(\mathbf{X}^{(L)}) \quad \text{with} \quad \mathbf{X}^{(0)} = \mathbf{X}, \tag{3}$$

$\mathbf{X}_j^{(\ell)} = \sigma(\text{normRows}(\widehat{\mathbf{A}}\mathbf{X}^{(\ell-1)})\mathbf{W}^{(\ell)})$, $\forall l \in [L]$, with $\widehat{\mathbf{A}} = [\mathbf{A}\ \mathbf{I}]$ (concatenation), normRows() normalizes each row to unit-norm, and $\theta = \{\mathbf{W}^{(\ell)}\}_{\ell \in [L]}$.

2.2 SUBGRAPH SAMPLING

In large-scale settings, where $\mathbf{A}$ contains large connected components, it is common to train on subgraph mini-batches. Let $\widetilde{\mathbf{A}} \in \mathcal{A}$ denote the adjacency corresponding to sampled subgraph. Some methods independently sample each edge (e.g., [Chen et al., 2018, Zou et al., 2019]). Other methods break-up the full adjacency into a number of smaller adjacency matrices as a pre-processing step [Chiang et al., 2019]. Finally, many methods employ tree-based sampling (§2.3).

[2] pre-processing adds self-loops and undirects graph.

By optimizing the GNN on samples $\widetilde{\mathbf{A}}$, in essence, these methods minimize

$$\min_{\theta} -\mathbb{E}_{\widetilde{\mathbf{A}}} \log p_\theta(\mathbf{Y} \mid \widetilde{\mathbf{A}}, \mathbf{X}). \tag{4}$$

2.3 TREE-BASED SUBGRAPH SAMPLING

A number of methods sample subgraphs as trees. Each tree is rooted at a labeled node $u \in \mathcal{V}_{\text{tr}}$ [*e.g.*, Hamilton et al., 2017, Markowitz et al., 2021, Ferludin et al., 2022]. Specifically, given node u and level-wise budgets $b^{(1)}, b^{(2)}, \dots b^{(h)} \in \mathbb{Z}_+$, these methods sample $b^{(1)}$ neighbors of u, and for each sampled $v \in \mathcal{N}(u)$, subsequently sample $b^{(2)}$ neighbors of v, and so on, until sampling depth h is reached. This process can be described by a recursive function:

Algorithm 1 Tree-based subgraph sampling of $\widetilde{\mathbf{A}}$

1: **Input:** Seed $u \in \mathcal{V}$; sampling budgets: $[b^{(1)}, \dots, b^{(h)}]$;
2: **Output:** Adjacency $\widetilde{\mathbf{A}}$ of sampled subgraph.
3: $\widetilde{\mathbf{A}} \leftarrow \mathbf{0}$ // initialize empty to populate recursively
4: **def** recursiveNeighbors(u', s):
5: **if** $s > h$: **return**
6: $\{v_k\}_{k=1}^{b^{(s)}} \sim Q(\mathcal{V} \mid u')$ // sample $b^{(s)}$ neighbors
7: **for each** v_k **do**
8: $\widetilde{\mathbf{A}}_{u',v_k} \leftarrow 1$
9: recursiveNeighbors(v_k, $s+1$)
10: recursiveNeighbors(u, 1)

$Q(\mathcal{V} \mid u)$ (line 6) can be hand-crafted. Popular sampling implementations [Hamilton et al., 2017, Ying et al., 2018, Markowitz et al., 2021, Ferludin et al., 2022] (implicitly) define Q as *i.i.d* $Q(\mathcal{V} \mid u) = \prod_{v \in \mathcal{V}} Q(v \mid u)$ and uniform $Q(v \mid u) = \frac{\mathbf{A}_{uv}}{\mathbf{D}_{uu}}$. Further, it is possible to design Q to assign probability as a function of node degrees, following Chen et al. [2018], Zeng et al. [2020], Zou et al. [2019].

3 SUBMIX

Our method can be seen as an extension of Tree-based subgraph sampling methods (§2.3, Algorithm. 1). Rather than assuming that the neighborhood distribution function Q is fixed apriori (see Line 6), we assume a tunable Q that is a convex combination other Q's that are known to perform well, e.g., uniform or as a function of node degrees. Formally,

$$Q_\phi(\mathcal{V} \mid u) = \prod_j \pi_\phi(j \mid u)\, Q_j(\mathcal{V} \mid u), \tag{5}$$

with categorical π_ϕ ($\sum_j \pi_\phi(j) = 1$ and $\pi_\phi(j) \geq 0\ \forall j$). j indexes *sampling heuristics*, e.g., $Q_{\text{RAND}}(\mathcal{V} \mid u) = \prod_{v \in \mathcal{V}} \frac{\mathbf{A}_{uv}}{\mathbf{D}_{uu}}$ corresponds to the choice of [Hamilton et al., 2017, Markowitz et al., 2021, Ferludin et al., 2022]. Other possible j choices are listed in §3.3.

In our construction, Q_j needs-not to factorize over neighbors. In particular, **we do not assume** *i.i.d* $Q_j(\mathcal{V} \mid u) = \prod_{v \in \mathcal{V}} Q_j(v \mid u)$. In other words, our algorithm accepts $Q_j(\mathcal{V} \mid u) \neq \prod_{v \in \mathcal{V}} Q_j(v \mid u)$ for which sampling an item from Q_j influences the subsequent probability of sampling remaining items. This allows for expressing sophisticated sampling heuristics that cannot be factorized per edge, for example based on maximum coverage, mutual information or other monotone submodular functions, a class of functions widely used for data sampling and summarization. For this class of functions, the well known Greedy algorithm (Nemhauser and Wolsey [1978], Minoux [2005], Mirzasoleiman et al. [2015]) produces an ordering in which every prefix of size b $(1 - 1/e)$-approximates the size-b subset of *maximum* objective value. Therefore, by producing an ordering based on such greedy algorithms we can capture a variety of data sampling methods with implicit $Q_j(\cdot \mid u)$.

3.1 MIXTURE SAMPLING ALGORITHM

Subgraph sampling algorithm that mixes heuristics can be written as:

Algorithm 2 Sample $\widetilde{\mathbf{A}}$ with heuristics

1: **Input:** Seed $u \in \mathcal{V}$; sampling budgets: $[b^{(1)}, \dots, b^{(h)}]$, J sampling heuristics $\{Q_j\}_{j \in [J]}$, mixture weights π_ϕ;
2: **Output:** Adjacency $\widetilde{\mathbf{A}}$ of sampled subgraph.
3: $\widetilde{\mathbf{A}} \leftarrow \mathbf{0}$ // initialize empty to populate recursively
4: **def** recursiveNeighbors(u', s):
5: **if** $s > h$: **return**
6: **for each** $j \in [J]$:
7: $b_j^{(s)} \leftarrow \pi_\phi(j \mid u') b^{(s)}$ // budget assigned to j
8: $\{v_k\}_{k=1}^{b_j^{(s)}} \sim Q(\mathcal{V} \mid u')$ // sample $b_j^{(s)}$ neighbors
9: **for each** v_k **do**
10: $\widetilde{\mathbf{A}}_{u',v_k} \leftarrow 1$
11: recursiveNeighbors(v_k, $s+1$)
12: recursiveNeighbors(u, 1)

Algorithm 2 is explained by-example in §3.4.

The training objective can then be written as:

$$\min_{\theta,\phi} \mathcal{L}(\theta, \phi) = \min_{\theta,\phi} -\mathbb{E}_{\widetilde{\mathbf{A}}} \log p_\theta(\mathbf{Y} \mid \widetilde{\mathbf{A}}, \mathbf{X}). \tag{6}$$

Note that the parameters ϕ determine the generated $\widetilde{\mathbf{A}}$ per Algorithm 2. In the remainder of this section, we discuss possible parameterizations of π_ϕ (§3.2), choices of Q_j (§3.3), and an approximation of Algorithm 2 that allows us to obtain analytical gradients of Eq. 6 w.r.t. ϕ (§3.4), *i.e.*, $\nabla_\phi \mathcal{L}$.

3.2 PARAMETERIZING WEIGHTS π_ϕ

We propose two versions for parameterizing π_ϕ, each making its own assumptions. The first version $\pi(\cdot \mid u)$ is in-

dependent of *how* u was sampled. In this case, (e.g., the $\pi(\cdot \mid u)$ is identical if u was a seed node or if it was recovered by any sampler j). The second version employs a Markov assumption: $\pi(\cdot \mid u)$ changes depeneding on the identity of the heuristic that gave rise to u (e.g., was u part of the seed batch? was it sampled with heuristic j?). The second (more general) version can be written as:

$$\pi_\phi(j \mid u \text{ is seed node}) = \text{softmax}(\mathbf{c})_j \stackrel{\text{def}}{=} c_j \tag{7}$$

$$\pi_\phi(j \mid u \text{ sampled by heuristic } k) = \text{softmax}(\mathbf{c}_k)_j \stackrel{\text{def}}{=} c_{jk} \tag{8}$$

where $\mathbf{c}, \mathbf{c}_1, \mathbf{c}_2, \ldots, \mathbf{c}_J \in \mathbb{R}^J$ are the trainable distribution parameters, and J denotes the number of components. In this case sampling parameters ϕ are:

$$\phi = \{\mathbf{c}, \mathbf{c}_1, \mathbf{c}_2, \ldots, \mathbf{c}_J\}. \tag{9}$$

When operating under the first version, $\pi_\phi(j \mid \text{seed } u) = \pi_\phi(j \mid \text{sampled } u) = \mathbf{c}$ and therefore $\phi = \{\mathbf{c}\}$. In general, N^{th} order Markov-chain would require $\mathcal{O}(J^{1+N})$ parameters.

3.3 PARAMETERIZING COMPONENTS Q_j

While component weights are trainable (§3.2), the components Q_j's are **fixed sampling heuristics**, listed next. The first four factorize on the neighbors, i.e., with $Q_j(\mathcal{V} \mid u) = \prod_{v \in \mathcal{V}} Q_j(v \mid u)$. As such, it suffices to write-down $Q_j(v \mid u)$. Subsequent Q_j's do not produce independent samples.

- Random sampler (Q_{RAND}): given node u, return subset of its neighbors $\mathcal{N}(u)$ uniformly at random.

$$Q_{\text{RAND}}(v \mid u) = \prod_{v \in \mathcal{V}} \frac{\mathbf{A}_{uv}}{\mathbf{D}_{uu}} \tag{10}$$

- Top-degree sampler (Q_{TOPDEG}): samples neighbor with probability increasing with its degree.

$$Q_{\text{TOPDEG}}(v \mid u) \propto \mathbf{A}_{uv} \log(\alpha + \delta_v). \tag{11}$$

  Here and below, we tried $\alpha = 1$ and $\alpha = 2$, without noticing any significant difference in performance. In practice, we tried sampling with and without replacement, and the results are almost identical.

- Least-degree sampler (Q_{LOWDEG}): samples neighbor with probability decreasing with its degree.

$$Q_{\text{LOWDEG}}(v \mid u) \propto \mathbf{A}_{uv} \log(\alpha + \delta_{\max} - \delta_v), \tag{12}$$

  where the maximum node degree $\delta_{\max} = \max_{u \in \mathbf{V}} \delta_u$.

- PageRank (Q_{PAGERANK}): samples neighbor with probability increasing with its PageRank [Page et al., 1999].

$$Q_{\text{PAGERANK}}(v \mid u) \propto \mathbf{A}_{uv} \log(\alpha + \text{PageRank}(v)). \tag{13}$$

- Self-sampler (Q_{SELF}) that samples only itself:

$$Q_{\text{SELF}}(v \mid u) = 1 \quad \textbf{iff} \quad v = u. \tag{14}$$

- Subset-based samplers, ones assigning probability to neighbor list, rather than independently per node, including:

$$Q_{\text{MAXCOVERX}}(\mathcal{V} \mid u),$$
$$Q_{\text{MAXCOVERA}}(\mathcal{V} \mid u),$$
$$\text{and} \quad Q_{\text{MAXCOVERAX}}(\mathcal{V} \mid u),$$

  for choosing edge-sets with destination nodes that are far-apart, respectively, in the feature-space, the graph topology, or a mixture of the two. For details see the §6.3.

Now that we have specified π_ϕ and Q_j, the next steps include determining how to obtain $\nabla_\phi \mathcal{L}$.

3.4 LEARNING ϕ WHEN SUBGRAPH SAMPLING

As formalized in Algorithm 2, tree-based subgraph sampling typically accepts:

- Seed node $u \in \mathcal{V}$, as discussed earlier.
- Sampling budgets $b^{(1)}, b^{(2)}, \cdots \in \mathbb{Z}_+$.

The algorithm then samples $b^{(1)}$ of u's neighbors, and for each of those neighbors, sample $b^{(2)}$ of its neighbors, and so on. The sampled edges are accumulated in $\widetilde{\mathbf{A}}$

For the sake of demonstration, consider only two samplers $[Q_{\text{LowDeg}}, Q_{\text{TopDeg}}]$ and a first hop sampling budget of $b^{(1)} = 20$. Also, suppose that the sampling budget is split with proportion $\pi = \mathbf{c} = [0.2, 0.8]$ (defined in Eq.7). Then, first-hop (direct) neighbors of u can be sampled as:

sample 4=$0.2(20) = c_1 \times b^{(1)} \triangleq b_1^{(1)}$ edges from Q_{LowDeg};

sample 16=$0.8(20) = c_2 \times b^{(1)} \triangleq b_2^{(1)}$ edges from Q_{TopDeg}.

In general, sampler Q_j gets a first-hop budget of:

$$b_j^{(1)} = c_j \times b^{(1)}. \tag{15}$$

Sampler Q_j can double its own $b_j^{(1)}$ by doubling the learnable $c_j = \text{softmax}(\mathbf{c})_j$, where $\mathbf{c} \in \phi$, *cf.* Eq.7. Subsequently, at the i^{th}-hop, when sampling edges for $u \in V^{(i)}$, Q_j receives budget

$$b_j^{(i)} = c_{jk} \times b^{(i)} \tag{16}$$

where $k \in \{1, \ldots, J\}$ is the index of mixture **mixture identity** from which u was sampled (*second* version, per §3.2).

In summary, training ϕ determines the partitioning of (integral[3]) budgets, among J samplers, which compete for budget via softmax transformation.

[3] $b_j^{(i)}$'s are not integral but one can do randomized rounding, *e.g.*, draw $\mathbf{Y} \sim \text{Bernoulli}(b_j^{(i)} - \lfloor b_j^{(i)} \rfloor)$ then $b_j^{(i)} \leftarrow \lfloor b_j^{(i)} + \mathbf{Y} \rfloor$.

3.4.1 How are $b_j^{(i)}$ utilized in GNN h?

Parameters ϕ influence $b_j^{(i)}$ and therefore $\widetilde{\mathbf{A}}$ that is consumed by the GNN (§2.1), upon which $\mathcal{L}$ is measured. To reason about how ϕ can be learned, we'd like to assess the influence[4] of $b_j^{(i)}$ on $\mathcal{L}$. One can express $\widetilde{\mathbf{A}}$ in terms of $b_j^{(i)}$. Let $e_j^{(i)}(u)$ be a permutation of $\mathcal{N}(u)$, assembled by repeatedly sampling from $\sim Q_j(\cdot \mid u)$. With this, we can define nodes $E^{(i)} \subseteq \mathcal{N}(i)$ sampled at depth i:

$$E^{(i)} = \bigcup_{j \in [J]} e_j^{(i)}(u)_{[:b_j^{(i)}]} \tag{17}$$

Where notation $[: b_j^{(j)}]$ selects the first $b_j^{(j)}$ of the ordered sequence. Entry (u, v) of the adjacency matrix can now be defined piece-wise:

$$\widetilde{\mathbf{A}}_{uv} = \begin{cases} 1, & \textbf{if } (u,v) \in \bigcup_{i \in [h]} E^{(i)} \\ 0, & \textbf{otherwise.} \end{cases} \tag{18}$$

The order of elements in $e_j^{(i)}(u)$ is always respected – only the first elements are chosen.

3.4.2 Backpropagation from p_θ to ϕ

GNN p_θ is a function of $\widetilde{\mathbf{A}}$. The information flows from ϕ to the GNN p_θ as $\phi \to \{b_j^{(i)}\}_{i,j} \to \{E^{(i)}\}_i \to \widetilde{\mathbf{A}} \to p_\theta$. Therefore, then it is natural to assess how to compute $\frac{\partial \mathcal{L}}{\partial \phi}$ from $\frac{\partial \mathcal{L}}{\partial p_\theta}$, *e.g.*, by backpropagation. However, this is not trivial. In particular, $\{E^{(i)}\}_i$ appear only in the branching condition for piecewise $\widetilde{\mathbf{A}}$ (Eq. 18). As such, $\widetilde{\mathbf{A}}$ is not continuously differentiable *w.r.t.* $b^{(i)} \in \mathbb{Z}_+$. Hence, the Jacobians $\left\{\frac{\partial A}{\partial \mathbf{c}}\right\}_{\mathbf{c} \in \phi}$ and therefore $\nabla_\phi \mathcal{L}$ are uninformative to compute.

We derive a continuous approximation $\dddot{\mathbf{A}}$ of $\widetilde{\mathbf{A}}$, such that the **values** of $\dddot{\mathbf{A}}$ are a function of b's and therefore $\mathbf{c}$'s. Moreover, $\dddot{\mathbf{A}}$ looks similar to piecewise $\widetilde{\mathbf{A}}$. Row $\widetilde{\mathbf{A}}_u^{(j)}$ of Eq. 18, when re-ordered by $e_j(u)$ can be plotted as a step function, see Fig. 2. To obtain a differentiable function, we propose to relax the stepwise selection with:

$$\dddot{\mathbf{A}}_{uv}^{(j)} = 1 - \sigma(\beta(r_{juv} - b_1^{(j)})), \tag{19}$$

where r_{juv} equals the position r for which $q_j(u)_r = v$, $\sigma(x) = (1 + e^{-x})^{-1}$, and β is a sharpness hyperparameter that we set $\beta = 2$. Re-ordering rows of $\dddot{\mathbf{A}}_{uv}^{(j)}$ according to $e_j(u)$ gives a reasonable approximation of the step function while being continuously-differentiable *w.r.t.* $\mathbf{c}$. See Fig. 2 for visual comparison between Eq. 18 & 19. Subsequently, we can calculate $\nabla_\phi \mathcal{L}$. Constructing $\dddot{\mathbf{A}}$ is detailed in Alg. 3.

[4] Influence should increase with $\frac{\partial \mathcal{L}}{\partial b_j^{(i)}}$.

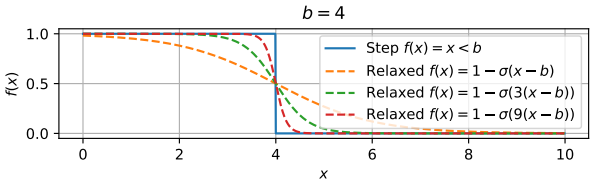

Figure 2: **Continuous Approximation**. For budget $b = 4$, solid line shows discrete step function, whereas dashed-lines show our continuous approximation (Eq. 19) each dashed line with a choice of sharpness constant β. As β increases, the continuous approximation becomes closer to the discrete step function.

Table 1: **Datasets**.

Dataset	Nodes	Edges
Citeseer	3,327 articles	4,732 citations
Cora	2,708 articles	5,429 citations
Pubmed	19,717 articles	44,338 citations
ogbn-arxiv	169,343 papers	1.2M citations
ogbn-products	2.5M products	61.9M co-purchases

4 EXPERIMENTAL EVALUATION

We conduct experiments using our method on five popular node classification datasets, summarized in Table 1. We utilize three datasets open-sourced by Yang et al. [2016]: **citeseer**, **cora** and **pubmed**, partially-popularized by Kipf and Welling [2017]. However, we use a larger training set (we include both training and validation into training) – as such we train our baselines. In addition, we use two datasets by OGB [Hu et al., 2020] for node classification: **ogbn-arxiv** and **ogbn-products**. For OGB datasets, we use the official train:validate:test partitions.

4.1 AGAINST ADVERSARIAL EDGE SOURCES

In these experiments, we assume that we have one $Q_{\text{NOISE}}(\cdot|u)$ that has uniform probability over all V, *i.e.*,

Table 2: Test accuracy of GNN models trained with **(left)** subgraphs sampled uniformly at random, against **(right)** subgraphs from clustered edge sources, per §4.2

	GCN($\widetilde{\mathbf{G}}$)	
Dataset	**baseline**: $\widetilde{\mathbf{A}} \sim Q_{\text{RAND}}$	**our-**§4.2: $\widetilde{\mathbf{A}} \sim Q_\phi$
Citeseer	74.08 ± 0.47	73.73 ± 0.50
Cora	84.74 ± 0.51	85.79 ± 0.64
Pubmed	83.90 ± 0.16	85.26 ± 0.29

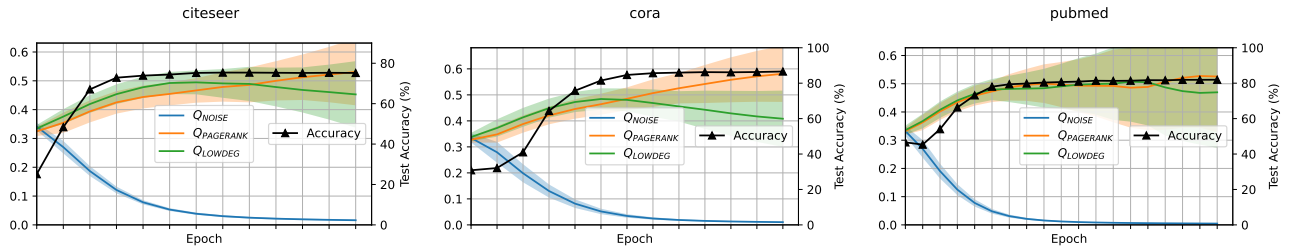

Figure 3: **Robust against noise edge sources**. Heuristic sampling negative edges gets ignored when learning ϕ (see §4.1).

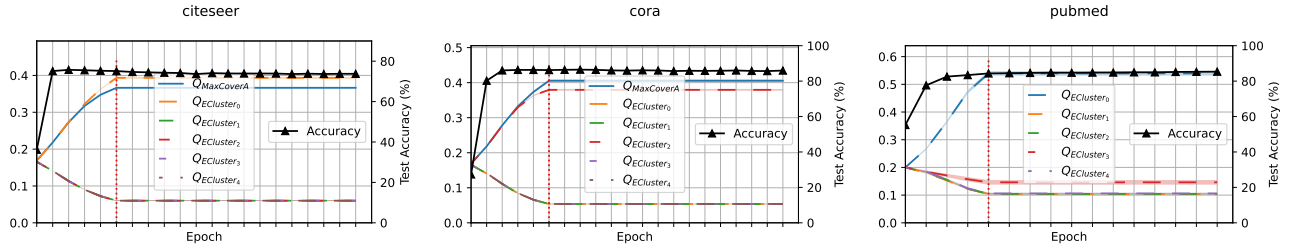

Figure 4: Q_ϕ distribution among edge clusters: each Q_{RAND} contains about $\frac{1}{5}$ of the edges. We freeze the sampling parameters ϕ after 25% of training epochs (red dashed line). For **pubmed**, it is sufficient to train on only $\frac{1}{5}$ of the edges while for **citeseer** and **cora** we supplemented with all graph edges but sampled according to greedy algorithm for max graph coverage, per §6.3.

Algorithm 3 Samples sparse $\ddot{\mathbf{A}}$ with continuous approximation

Input: Seed batch: $V^{(0)} \subseteq \mathcal{V}$; sampler heuristics $\{q_j\}_{j \le J}$; per-step sampling budgets: $[b^{(1)}, \dots, b^{(h)}]$;
Output: Adjacency $\ddot{\mathbf{A}}$ of sampled subgraph.
// Pair each $u \in V^{(0)}$ with producing component:
$\widetilde{V}^{(0)} \leftarrow [(\texttt{null}, u)$ **for** $u \in V^{(0)}]$
for $i \leftarrow 1$ **to** S **do**
  $\widetilde{V}^{(i)} \leftarrow [\,]$
  for (k, u) **in** $\widetilde{V}^{(i-1)}$ **do**
    for $j \leftarrow 1$ **to** J **do**
      for $r \leftarrow 1$ **to** b_i **do**
        $v \leftarrow q_j(u)[r]$
        $\widetilde{V}^{(i)}.\texttt{append}((j, v))$
        $b_j^{(i)} \leftarrow c_{kj} \times b^{(i)}$
        $\ddot{\mathbf{A}}_{uv} \leftarrow \ddot{\mathbf{A}}_{uv} + 1 - \sigma(\beta(r - b_i^{(j)}))$

it is very likely that $v \notin \mathcal{N}(u)$ for $v \sim Q_{\text{NOISE}}(\cdot|u)$. Since Q_{NOISE} favors negative edges, we would hope that our learning algorithm is able to down-weigh Q_{NOISE}. Specifically, we hope to learn $\phi = \{\mathbf{c}\}$ with $c_{\text{NOISE}} \approx 0$.

Setup: We train GCN of [Kipf and Welling, 2017], with training examples being subgraphs sampled according to §3.1&§3.4, with sampling budgets $b^{(1)} = 20, b^{(2)} = 10$. We use: $\{Q_j\}_j = \{Q_{\text{PAGERANK}}, Q_{\text{LOWDEG}}, Q_{\text{NOISE}}\}$. We attempt Datasets: **citeseer**, **cora**, **pubmed**.

Discussion: We see that, as summarized in Fig. 3, subgraph sampling parameters ϕ learn to ignore edges coming from Q_{NOISE}. In all cases, c_{NOISE} goes towards 0. We run each experiment ten times, and plot the average Q_ϕ through time, shading up-to the standard deviation.

Hyperparameters: We use GCN model of [Kipf and Welling, 2017] with default hyperparameters, *i.e.*, two GCN layers, hidden dimension of 32, trained with dropout and Adam optimizer using learning rate of 0.01 (but 0.005 for **pubmed**). The learning rate for ϕ parameters is set to 0.05.

4.2 EMPHASIZING FAVORITE EDGES

We test if our method can choose edges that are preferred for the node classification task. Specifically, we:

1. Partition edges $\mathbf{E}$ into $\mathbf{E}_1^{\text{p}}, \mathbf{E}_2^{\text{p}}, \mathbf{E}_3^{\text{p}}, \mathbf{E}_4^{\text{p}}, \mathbf{E}_5^{\text{p}}$ – we experiment only with 5 partitions. The partition is based on the features of edge endpoints (details below).
2. We instantiate Q_{ECLUSTER_m} for partition $\mathbf{E}_m^{\text{p}}$ with:
$$Q_{\text{ECLUSTER}_m}(v \mid u) = \frac{1}{|(\tilde{v}, u) \in \mathbf{E}_m^{\text{p}}|} \mathbb{1}[(u, v) \in \mathbf{E}_m^{\text{p}}],$$
where $\mathbb{1}[\cdot]$ is the indicator function, evaluating to 1 if its argument is true and otherwise to 0. Our learning algorithm should pick among the 5 edge clusters: if $\phi = \{\mathbf{c}\}$ was learned, such that, $c_m > c_\ell$, then, $\mathbf{E}_m^{\text{p}}$ contains **better edges** than $\mathbf{E}_\ell^{\text{p}}$, according to the objective function.

Partitioning algorithm: For each edge $(u, v) \in \mathbf{E}$, calculate

edge feature $\mathbf{X}_{vu} = \mathbf{X}_{uv} \in \mathbb{R}^d$ as the Hadamard product of node features: $\mathbf{X}_{uv} = \mathbf{X}_u \circ \mathbf{X}_v$. Then, we run k-means on $\{\mathbf{X}_{uv}\}_{(u,v)\in\mathbf{E}}$ with $k = 5$. If k-means assigns cluster m to feature vector $\mathbf{X}_{uv}$, then edge (u, v) will be in $\mathbf{E}^{\text{p}}_m$. Note that $\mathbf{E} = \cup_m \mathbf{E}^{\text{p}}_m$ and that $\mathbf{E}^{\text{p}}_m \cap \mathbf{E}^{\text{p}}_\ell = \emptyset$ for all $m \neq \ell$.

Datasets and Hyperparameters: same as §4.1.

Setup We compare two GCNs, both of them are trained with sampling budgets $b^{(1)} = 20, b^{(2)} = 10$. The first GCN is **baseline**, where training examples are subgraphs sampled uniformly at random *i.e.*, only Q_{ECLUSTER} is used: given a seed node, a random subtree is sampled by recursively choosing from neighbors with equal probability. The second GCN is **our-**§4.2, where training examples are subgraphs sampled according to §3.1&§3.4. For **pubmed**, we use $\{Q_j\}_j = \{Q_{\text{RAND}^{\text{p}}_m}\}_{m=1}^5$. For **cora** and **citeseer**, we additionally use $Q_{\text{MAXCOVERA}}$, which samples edges due to greedy algorithm, as detailed in §6.3.

Discussion: According to Table 2, we see that over-emphasizing edges, based on the cluster of the endpoint features, helps for two of the three datasets. In addition, Fig. 4 summarizes the learned $\mathbf{c} \in \phi$. We see that each graph indeed chooses some edges to be over-repeated (assigning remainder edge sources to ≈ 0). For **pubmed**, we can get reasonable results while **ignoring most graph edges**, *i.e.*, ϕ learns to discard all but one $\mathbf{E}^{\text{p}}$.

5 MELD INTO DECOUPLED GNN

Many methods train node-wise multi-layer perceptron $\text{MLP}(\mathbf{X}) : \mathbb{R}^d \to \mathbb{R}^z$ mapping node features into classes, where the trainable transformation does **not** use the graph. Instead, the graph is used in a non-trainable **propagation** [Wu et al., 2019, Frasca et al., 2020, Gasteiger et al., 2019a, Duan et al., 2022]. Most recently, EnGCN [Duan et al., 2022] proposes to run the (full-graph) propagation only a few times during training, e.g., once every 100 epochs on node-wise MLP. Our generative framework can be utilized in decoupled GNNs. Since the **propagation** step is allowed to consider the full graph, we define a full-graph adjacency $\widetilde{\mathbf{A}} \in \mathbb{R}^{|V|\times|V|}$, based on §3, as:

$$\widetilde{\mathbf{A}}_{uv} = \mathbb{E}_{j\sim\pi_\phi}\left[Q_j(v \mid u)\right] \triangleq \mathbb{E}_{j\sim\pi_\phi}\left[\mathbf{A}_{(j)uv}\right] \tag{20}$$

where $\mathbf{A}_{(j)}$ reshapes Q_j into a matrix.

We extend EnGCN [Duan et al., 2022] with our framework. We use our ϕ-parameterized $\widetilde{\mathbf{A}}$ instead of their proposed propagation matrix of $\widehat{\mathbf{A}}$ – Gasteiger et al. [2019b,a] show other plausible choices for propagation matrix. EnGCN uses propagation as a linear operator:

$$\mathbf{X}^{(0)} \leftarrow \mathbf{X}; \qquad \mathbf{X}^{(i+1)} \leftarrow \widetilde{\mathbf{A}}\mathbf{X}^{(i)} \tag{21}$$

Naïvely feeding $\mathbf{X}^{(i)}$'s (Eq. 21) into MLP GNN, unfortunately, would cast the decoupled GNNs into graph-consuming variants, as one would have to backpropagate through the adjacancy values to get $\nabla_\phi \mathcal{L}$. Nonetheless, we maintain decoupling by considering linearity of expectation and of the propagation operation.

$$\mathbf{X}^{(i+1)} = \widetilde{\mathbf{A}}\mathbf{X}^{(i)} = \mathbb{E}_{j\sim Q_\phi}\left[\mathbf{A}_{(j)}\mathbf{X}^{(i)}\right]. \tag{22}$$

We then batch-compute: $\mathbf{A}_{(1)}\mathbf{X}^{(i-1)}, \ldots, \mathbf{A}_{(J)}\mathbf{X}^{(i-1)}$. For learning layer (i) and node-wise MLP : $\mathbb{R}^{z_i} \to \mathbb{R}^{z_{i+1}}$, the input features of MLP can be set to:

$$\mathbf{X}^{(i)} = \sum_{j\leq J} \mathbf{A}_{(j)}\mathbf{X}^{(i-1)} c_j \tag{23}$$

by reading batch-computed values. Now, obtaining $\nabla_\phi \mathcal{L}$ becomes trivial as $\mathbf{c}$ entries appear in the summation. Yet, the graph transformation (left-multiplying by $\mathbf{A}_{(j)}$) happens in a fixed process, maintaining decoupling.

Unfortunately, we incur a cost for increasing the size of the latent space (linear in $\mathcal{O}(J)$). However, for inference *i.e.*, once $\phi = \{\mathbf{c}\}$ is learned and fixed, one need only store one vector per example.

5.1 EVALUATING OUR DECOUPLED GNNS

Datasets. OGB of Hu et al. [2020] open-source many graph tasks. We run our methods on node classification tasks (OGBN), specifically, homogeneous[5] ones: **ogbn-arxiv** and **ogbn-products**. Baselines. We copy baseline numbers from the OGBN public leaderboard[6]. We choose compare EnGCN variants: (a) using external data and (b) not using external data. Table 3 summarizes the results.

Our model. we modify the code of EnGCN [Duan et al., 2022] to import our contribution. We replace their propagation operation, as described in this section.We inherit all their hyper-parameters, as detailed in [Duan et al., 2022].

Table 3: **Node Classification for OGBN datasets**. Method (a) use external data: text of abstract or product description, resp., for ogbn-arxiv and ogbn-products. Methods (b) use only provided feature vectors. Last line is ours.

	Method	ogbn-arxiv	ogbn-products
(a)	GLEM+EnGCN	79.66 ± 0.06	90.14 ± 0.12
(b)	EnGCN	77.98 ± 0.07	87.98 ± 0.04
	Ours: EnGCN($\widetilde{\mathbf{A}}$)	82.26 ± 0.09	88.59 ± 0.07

[5]Extending to heterogeneous settings is left as future work.

[6]After the UAI submission deadline, it was revealed that EnGCN codebase accidently uses labels in the validation partition: https://github.com/VITA-Group/Large_Scale_GCN_Benchmarking/issues/5#issue-1597789310. As such, EnGCN methods were removed from the leaderboard.

6 MISCELLANEOUS

6.1 COMPUTATIONAL EFFICIENCY

In our experiments, updating parameters ϕ introduces a 3x slowdown. To work around the slowdown, we freeze changing the ϕ parameters sometime through training, e.g., after $\frac{1}{4}$'th of the total training epochs.

6.2 REPRESENTATIONAL CAPACITY

It is worth noting learning the sampling parameters has a higher representational capacity compared to fixed heuristics. Specifically, our method should be able to recover the sampling process of [Hamilton et al., 2017, Ying et al., 2018, Markowitz et al., 2021, Ferludin et al., 2022] if ϕ learns to assign all its mass on Q_{RAND} by learning $\pi_{\text{RAND}} \approx 1$.

6.3 GREEDY-BASED Q

We now sampling heuristics $Q_{\text{MAXCOVERX}}$, $Q_{\text{MAXCOVERA}}$, and $Q_{\text{MAXCOVERAX}}$. These heuristics are not sampled *i.i.d*. Specifically, when sampling list of neighbors for node u from $Q_{\text{MAXCOVER}^*}(\mathcal{V} \mid u)$, then sampling some node $k \in \mathcal{V}$ will influence the probability of subsequent nodes to be sampled. In particular, sampling node $k \in \mathcal{V}$ will encourage that subseqently-sampled nodes should be far from k.

- $Q_{\text{MAXCOVERX}}$: given a node, prefers neighbors with complimentary features. Specifically, suppose sampled nodes
$$E_u \sim Q_{\text{MAXCOVERX}}(\mathcal{V} \mid u) \tag{24}$$
contains nodes $E_u = \{v_1, v_2, v_3, \dots\} \subseteq \mathcal{N}(u)$, then, with high probability, $v_1, v_2, v_3, \dots$ are far-apart, in the feature space. In other words, the pairwise Euclidean distances
$$||\mathbf{X}_v - \mathbf{X}_{v'}||, \ \ \forall (v \neq v') \in E_u, \tag{25}$$
are large.

  While the Q_j's in §3.3, nodes can be sampled independently, the MAXCOVERX heuristics are necessarily set-based, *i.e.*, the inclusion of one node affects the probability of sampling another.

- $Q_{\text{MAXCOVERA}}$: prefers neighbors with complimentary structural positions. Specifically, sampling $E_u \sim Q_{\text{MAXCOVERA}}(\mathcal{V} \mid u)$ yields edges $E_u = \{v_1, v_2, v_3, \dots\} \subseteq \mathcal{N}(u)$, such that, the Euclidean distances
$$||\mathbf{U}_v - \mathbf{U}_{v'}||, \ \ \forall (v \neq v') \in E_u, \tag{26}$$
are large, where $\mathbf{U}$ is the matrix of orthonormal eigenvectors of $\widehat{\mathbf{A}}$.

- $Q_{\text{MAXCOVERAX}}$: prefers neighbors with complimentary neighborhood features. Similar to the above two – sampling $E_u \sim Q_{\text{MAXCOVERA}}(\mathcal{V} \mid u)$ yields edges $E_u = \{v_1, v_2, v_3, \dots\} \subseteq \mathcal{N}(u)$, such that, the Euclidean distances
$$||(\widehat{\mathbf{A}}\mathbf{X})_v - (\widehat{\mathbf{A}}\mathbf{X})_{v'}||, \ \ \forall (v \neq v') \in E_u, \tag{27}$$
are large.

The MAXCOVER heuristics prefer neighbors that increase diversity. Sampling from $Q_{\text{MAXCOVERX}}(\cdot \mid u)$ should yield neighbors with substantially different features. Sampling from $Q_{\text{MAXCOVERA}}(\cdot \mid u)$ should yield neighbors $E_u \subseteq \mathcal{N}(u)$, such that, E_u has maximum vertex cover. Finally, sampling from $Q_{\text{MAXCOVERAX}}(\cdot \mid u)$ should yield b neighbors with substantially different neighboring features.

In general, algorithms for selecting subset that maximize coverage are NP-hard. However, there are many greedy approximations. Our greedy approximation is as follows:

1. (**Define** H) For MAXCOVERX, let $\mathbf{H} \leftarrow \mathbf{X}$. For MAXCOVERAX, let $H \leftarrow \widehat{\mathbf{A}}\mathbf{X}$. Finally, for MAXCOVERA, let $\mathbf{H} \leftarrow \mathbf{U}\Lambda^{\frac{1}{2}}$ were orthonormal matrix U contain the eigenvectors of $\widehat{\mathbf{A}}$ and diagonal matrix Λ contain eigenvalues.
2. (**Cluster** H) Run k-means algorithm on rows of H.
3. (**Round-robin**) Sampling from $Q_{\text{MAXCOVER}^*}(E \mid u)$ should yield target nodes that round-robin the clusters.

7 RELATED WORK

We broadly summarize methods from our directions of interest: subgraph sampling and decoupled GNNs.

Subgraph Sampling: Chiang et al. [2019] partition input graphs into subgrahs using algorithm for graph clustering such as METIS [Karypis and Kumar, 1998]. The subgraphs are computed once and re-used for training. Other methods sample graphs on-the-fly. For instance, Chen et al. [2018] sample nodes independently and an edge is only included if both of its endpoints are sampled. Zou et al. [2019] then propose to do conditional sampling: nodes sampled at $V^{(i)}$ would influence nodes sampled at $V^{(i+1)}$. This conditional assumption is also implied by recursive subgraph sampling, such that of [Hamilton et al., 2017, Ying et al., 2018, Markowitz et al., 2021, Ferludin et al., 2022]. While we fit into the last mentioned models, as our sampling is also tree-based, in addition, we learn $\mathrm{P}(v \mid u)$ for all $(u, v) \in \cup_i E^{(i)}$, as convex combination of $\{Q_j(v \mid u)\}_{j \leq J}$. This allows our method to perform online edge subset selection, where subsets can be pre-computed by clustering.

Learnable sampling. Yoon et al. [2021] also propose to learn sampling parameters using gradients. However, they assume (the original) discrete sampling process and they learn using reinforcement-learning policy gradients. Instead, we obtain analytical gradients by relaxing the discrete process. Wang et al. [2021] also trains a sampler. However, they parameterize their sampler in terms of features. In our case, our

sampler mixes a number of heuristics. Our heuristics are based on structural properties (e.g., node degree or PageRank) rather than features.

Decoupled GNNs: One of the earliest decoupled GNNs is SimpleGCN [Wu et al., 2019]. Before learning starts, graph propagation is applied as a pre-processing step. Learning only use the graph-transformed information but without using the graph. On the other hand, Gasteiger et al. [2019a] also decouples the learned parameters from the graph propagation, however, by applying the propagation after the node-level model. While these two mentioned methods apply the fixed graph transformation once (after or before node-level model), EnGCN [Duan et al., 2022] interleaves node-level learning with graph propagation. In this work, we modify EnGCN by replacing its fixed propagation function, by a function that conducts a variety of propagations $\{\mathbf{A}_{(j)}\}$ in parallel, each propagation j according to heuristic $j \in [J]$. We learn convex combination of precomputed J graph-transformed features.

For space constraints, we do not discuss other directions for scaling-up learning of GNNs, including utilizing distributed compute [Lerer et al., 2019] or using historical embeddings [Chen et al., 2017, Fey et al., 2021].

8 CONCLUSION

In this work we introduced a method for learning graph sampling while training Graph Neural Networks (GNNs). Our proposed method parameterizes graph sampling as a convex combination of different heuristics. We apply our method in two popular regimes for learning GNNs: (D.1) training on sampled subgraphs, and (D.2) *decoupled GNN*: graph propagation step is free from trainable parameters. For (1), we propose a continuous approximation of the discrete process. We evaluate our method in three scenarios. (i) our method can learn to discard edge sources that are noisy; (ii) if edges are clustered using their endpoint features, then our method can learn favorite edge clusters; finally, (iii) integrating our method with decoupled GNNs (D.2) achieves SOTA results on ogbn-arxiv and ogbn-products.

References

Peter W. Battaglia, Jessica B. Hamrick, Victor Bapst, Alvaro Sanchez-Gonzalez, Vinícius Flores Zambaldi, Mateusz Malinowski, Andrea Tacchetti, David Raposo, Adam Santoro, Ryan Faulkner, Çaglar Gülçehre, H. Francis Song, Andrew J. Ballard, Justin Gilmer, George E. Dahl, Ashish Vaswani, Kelsey R. Allen, Charles Nash, Victoria Langston, Chris Dyer, Nicolas Heess, Daan Wierstra, Pushmeet Kohli, Matthew M. Botvinick, Oriol Vinyals, Yujia Li, and Razvan Pascanu. Relational inductive biases, deep learning, and graph networks. *arxiv/1806.01261*, 2018.

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

Ines Chami, Sami Abu-El-Haija, Bryan Perozzi, Christopher Ré, and Kevin Murphy. Machine learning on graphs: A model and comprehensive taxonomy. In *Journal on Machine Learning Research*, 2022.

Jianfei Chen, Jun Zhu, and Le Song. Stochastic training of graph convolutional networks with variance reduction. *arXiv preprint arXiv:1710.10568*, 2017.

Jie Chen, Tengfei Ma, and Cao Xiao. FastGCN: Fast learning with graph convolutional networks via importance sampling. In *International Conference on Learning Representations*, 2018.

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

Keyu Duan, Zirui Liu, Peihao Wang, Wenqing Zheng, Kaixiong Zhou, Tianlong Chen, Xia Hu, and Zhangyang Wang. A comprehensive study on large-scale graph training: Benchmarking and rethinking. In *Thirty-sixth Conference on Neural Information Processing Systems Datasets and Benchmarks Track*, 2022.

Oleksandr Ferludin, Arno Eigenwillig, Martin Blais, Dustin Zelle, Jan Pfeifer, Alvaro Sanchez-Gonzalez, Sibon Li, Sami Abu-El-Haija, Peter Battaglia, Neslihan Bulut, et al. TF-GNN: Graph neural networks in tensorflow. *arXiv preprint arXiv:2207.03522*, 2022.

M. Fey, J. E. Lenssen, F. Weichert, and J. Leskovec. GNNAutoScale: Scalable and expressive graph neural networks via historical embeddings. In *International Conference on Machine Learning (ICML)*, 2021.

Fabrizio Frasca, Emanuele Rossi, Davide Eynard, Benjamin Chamberlain, Michael Bronstein, and Federico Monti. Sign: Scalable inception graph neural networks. In *ICML 2020 Workshop on Graph Representation Learning and Beyond*, 2020.

Johannes Gasteiger, Aleksandar Bojchevski, and Stephan Günnemann. Combining neural networks with personalized pagerank for classification on graphs. In *International Conference on Learning Representations*, 2019a.

Johannes Gasteiger, Stefan Weißenberger, and Stephan Günnemann. Diffusion improves graph learning. In *Conference on Neural Information Processing Systems*, 2019b.

W. Hamilton, R. Ying, and J. Leskovec. Inductive representation learning on large graphs. In *Advances in Neural Information Processing Systems*, 2017.

Weihua Hu, Matthias Fey, Marinka Zitnik, Yuxiao Dong, Hongyu Ren, Bowen Liu, Michele Catasta, and Jure Leskovec. Open graph benchmark: Datasets for machine learning on graphs. In *arXiv*, 2020.

George Karypis and Vipin Kumar. A fast and high quality multilevel scheme for partitioning irregular graphs. *SIAM Journal on Scientific Computing*, 20(1):359–392, 1998.

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

Adam Lerer, Ledell Wu, Jiajun Shen, Timothee Lacroix, Luca Wehrstedt, Abhijit Bose, and Alex Peysakhovich. Pytorch-biggraph: A large-scale graph embedding system. In *The Conference on Systems and Machine Learning*, 2019.

Elan Sopher Markowitz, Keshav Balasubramanian, Mehrnoosh Mirtaheri, Sami Abu-El-Haija, Bryan Perozzi, Greg Ver Steeg, and Aram Galstyan. Graph traversal with tensor functionals: A meta-algorithm for scalable learning. In *International Conference on Learning Representations*, ICLR, 2021.

Michel Minoux. Accelerated greedy algorithms for maximizing submodular set functions. In *Optimization Techniques: Proceedings of the 8th IFIP Conference on Optimization Techniques Würzburg, September 5–9, 1977*, pages 234–243. Springer, 2005.

Baharan Mirzasoleiman, Ashwinkumar Badanidiyuru, Amin Karbasi, Jan Vondrák, and Andreas Krause. Lazier than lazy greedy. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 29, 2015.

Federico Monti, Michael M. Bronstein, and Xavier Bresson. Geometric matrix completion with recurrent multi-graph neural networks. In *International Conference on Machine Learning*, 2017.

George L Nemhauser and Laurence A Wolsey. Best algorithms for approximating the maximum of a submodular set function. *Mathematics of operations research*, 3(3): 177–188, 1978.

Lawrence Page, Sergey Brin, Rajeev Motwani, and Terry Winograd. The pagerank citation ranking: Bringing order to the web. Technical report, Stanford InfoLab, 1999. URL `http://ilpubs.stanford.edu:8090/422/`.

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

Yu Wang, Zhiwei Liu, Ziwei Fan, Lichao Sun, and Philip Yu. Dskreg: Differentiable sampling on knowledge graph for recommendation with relational gnn, 08 2021.

Felix Wu, Amauri Souza, Tianyi Zhang, Christopher Fifty, Tao Yu, and Kilian Weinberger. Simplifying graph convolutional networks. In *International Conference on Machine Learning*, 2019.

Zhilin Yang, William W Cohen, and Ruslan Salakhutdinov. Revisiting semi-supervised learning with graph embeddings. In *International Conference on Machine Learning*, 2016.

Rex Ying, Ruining He, Kaifeng Chen, Pong Eksombatchai, William L. Hamilton, and Jure Leskovec. Graph convolutional neural networks for web-scale recommender systems. In *ACM SIGKDD international conference on Knowledge discovery and data mining*, 2018.

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*, KDD '21, pages 2046–2056, 2021.

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

Difan Zou, Ziniu Hu, Yewen Wang, Song Jiang, Yizhou Sun, and Quanquan Gu. Few-shot representation learning for out-of-vocabulary words. In *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems, NeurIPS*, 2019.