GraphSAID: *Graph S*ampling via *A*ttention based *I*nteger Programming Metho*d*

Ziqi Liu
New York University Center for Data Science
New York, United States
ziqiliu@nyu.edu

Laurence Liu
FCC Analytics
Hong Kong, China
laurence0636@outlook.com

ABSTRACT

Graphs are extensively employed in data mining and machine learning owing to their remarkable ability to model real-world objects and their relationships. However, as graphs scale up, they present several challenges. To tackle these issues, graph sampling methods have gained popularity by selecting a representative subgraph within a given budget. However, most graph sampling approaches rely on graph-structure information and cannot simultaneously consider node feature interaction and selection bias to perform graph sampling. Given the recent success of the attention mechanism in model training, it is worth investigating its potential to enhance graph sampling methods and overcome their challenges. The primary objective of this work is to establish a novel connection between the learned attention and the graph sampling problem using the Integer Programming method. To accomplish this, we propose a novel solution, *GraphSAID*, which utilizes an attention learning stage to generate initial node-level attention, followed by an aggregation stage to compute connected component scores that are independent of the budget. Finally, the Integer Programming method is employed to optimize an objective function that considers both the budget value and the user-defined selection bias. Empirical results on 1 synthesized and 3 real-world graph datasets demonstrate its superior performance. Additionally, we showcase the ease with which selection bias (user control) can be incorporated into *GraphSAID* to further improve performance.

KEYWORDS

Graph Sampling; Attention; Graph Learning; Deep Learning; Subgraph Generation

ACM Reference Format:
Ziqi Liu and Laurence Liu. 2024. *GraphSAID*: *Graph S*ampling via *A*ttention based *I*nteger Programming Metho*d*. In *Proc. of the 23rd International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2024), Auckland, New Zealand, May 6 – 10, 2024*, IFAAMAS, 9 pages.

1 INTRODUCTION

In recent years, the increasing availability of large-scale graph data in various applications has raised the importance of graph data study, particularly in the areas of data mining and machine learning. However, many real-world networks are large and complex, making it difficult to analyze or manipulate the entire network for humans or agents. One may consider using graph sampling methods, which select representative subgraphs (subsets of nodes or edges) from the network that can further facilitate network analysis, visualization, modeling, and deployment [4, 5, 15, 29, 37, 45, 47, 48, 62]. For example, a sufficiently small yet effective subgraph can further compress the model architecture to facilitate inference [47], mitigate the deleterious effects of data noise for downstream analyses [37], and expedite the hyperparameter tuning process [4].

The common approach to performing graph sampling in large-scale graph datasets [35, 44, 53] relies on node-level traversal mechanisms such as [41, 48], which perform random-walk based on node degrees. However, traversal node by node results in low data efficiency and a lack of global structure information such as community [45], which can be deleterious when performing clustering analysis on the sampled subgraph [15]. Therefore, existing methods in graph sampling focus more on a higher level (e.g., neighbor exploration). They combine both breadth-first search (BFS) and depth-first search (DFS) [14, 18, 31], or community expansion to seek new nodes such that the sampled set can reach the largest number of unknown nodes [34]. Recent works [38, 56] focus on higher-level geometry metrics such as Ollivier-Ricci Curvature [43] to better capture the community clustering information by uniformizing the edge Ricci curvatures. However, sampling a subgraph solely depending on node- or community-level geometry features results in a lack of graph semantics, which is more severe if the sampled subgraph is used in downstream learning tasks such as pre-trained or teacher networks [47, 58].

Graph neural networks (GNN) have been extensively employed for tasks involving node- or edge-level classification tasks [8, 20], graph representation learning [7, 21] and generative learning [32]. The literature on GNN [57, 60, 65] reveals the ability of deep models to learn graph semantics using message passing mechanism [65] or graph convolutional kernels [54]. In particular, [20] has proven that a GNN with two consecutive graph convolutional layers can predict both a node label and its clustering coefficient [52]. In order to integrate graph semantics into the graph sampling process, it is encouraging to employ a learned GNN as a semantic extractor carrying semantic information inherent in the graph.

To this end, in this work, we propose *Graph Sampling via Attention Based Integer Programming Method (GraphSAID)*, a novel three-stage approach to perform graph sampling. Specifically, in the first stage, we use an *Attended Edge Rank Calculation* method, which employs *attention*, the learned graph neural network by-product to assign ranks to each edge in the graph, indicating their relative importance within the graph. Secondly, we use an *Attended Connected Component Generation* method to aggregate the edge rank scores

Proc. of the 23rd International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2024), N. Alechina, V. Dignum, M. Dastani, J.S. Sichman (eds.), May 6 – 10, 2024, Auckland, New Zealand. © 2024 International Foundation for Autonomous Agents and Multiagent Systems (www.ifaamas.org).

into connected component scores, which are subsequently merged to form the final selected subgraph. Finally, we employ an *Integer Programming Based Node Selection* method to efficiently select a subgraph for a given budget. This is achieved by modeling the subgraph selection problem as an Integer Programming optimization problem, using the connected component scores obtained from the previous stages.

We test our solution on four datasets - one synthesized dataset and three real-world datasets [35, 44, 53] with diverse graph properties. Our findings demonstrate that our approach outperforms graph-structure-based sampling methods in both subgraph effectiveness and downstream training convergence. Furthermore, our solution's flexibility is highlighted through case studies, which reveals its ability to seamlessly integrate selection bias (user control) to aid the subgraph selection.

Our contributions can be summarized in fourfold:

- To our best knowledge, this is the first work using learned attention as a by-product to perform a graph sampling process, which incorporates graph semantics leading to a more effective subgraph.
- Since the raw learned attention is local, we propose two stages, *Attended Edge Rank Calculation* and *Attended Connected Component Generation*, to shift the local attention to cluster level, which significantly facilitates the overall sampling efficiency.
- Our novel *Integer Programming Based Node Selection* treats a graph sampling process as a connected components selection problem, which balances both local and global, geometry and semantic information for the output subgraph.
- We demonstrate the ease of incorporating user control in the graph sampling, and empirically show that the generated subgraph by *GraphSAID* can facilitate both downstream training efficiency and the subgraph effectiveness.

2 RELATED WORK

Our algorithm is conceptually rooted in graph sampling techniques, as well as in recent advancements in graph neural networks and the implementation of sophisticated attention mechanisms.

2.1 Graph Sampling

There exist two primary strategies for graph sampling methods. One is random sampling (e.g., [42, 48]), by which each node is sampled according to a probability assigned. [42] demonstrates that the larger the spectral gap of the Random Walk transition matrix, the less likely that a random walker will get stuck. This finding provides additional support for promising research directions, including [10, 17, 41]. Random Walk with Fly-back probability (RWF) introduces a fly-back probability p during each iteration of the random walk, where the walk will go back to the initial node with probability p. [30] and [24] refered to RWF as "random jump". The second strategy is exploration-based (e.g., [18, 31]), in which it begins by selecting a node at random and subsequently exploring its neighbors in later iterations. Snow Ball (SB) [18] is commonly used in investigations of hidden populations. In each round, an entity is required to refer to a fixed number of related entities. Forest Fire (FF) was proposed as a graph generation model to simulate social network properties [31]. [30] adapted FF for graph sampling, which is an extension of SB, except that in each iteration, the number of neighbors sampled is probabilistic. Unlike these previous approaches, we leverage the learned attention information in order to capture the prior knowledge before sampling the graph. A more recent method [56], built upon the motivation that between-community Ollivier Ricci curvature (ORC) is larger than within-community ORC, proposed a greedy framework to sample nodes from all communities to the subgraph. However, how much graph semantics can ORC embed is unclear.

2.2 Graph Neural Networks.

Research has explored information diffusion through edges as a means of leveraging graph typologies for learning. Extended from RNN, GNN [46] recursively updates node features till equilibrium to allow learning on more generalized graphs, e.g. acyclic, cyclic, directed, or undirected. GCN [26] summarized and simplified previous works (e.g., [5, 12, 22]) by proposing a layer-wise linear model that restricts information propagation to first-order neighbors only. While such simplification largely reduces computational costs, spectral-based GCNs are still computationally expensive as they by nature require the eigen decomposition of the Laplacian matrix. On the other hand, GraphSAGE [20] or its variants [8, 9, 61] are powerful graph neural network models that learn node representations by aggregating information from the node's local graph neighborhood using a flexible and scalable sampling strategy, which greatly reduces the in-memory computation cost. These neural network designs enable efficient computation of prior knowledge from large graphs, which can further be leveraged to perform downstream tasks such as graph sampling.

2.3 GNNs meet Attention.

The attention mechanism [59] has emerged as a powerful tool in deep neural networks for enabling the model to selectively focus on important features of the input data. By assigning weights to different parts of the input, the attention mechanism allows the network to attend to the most relevant information, improving both the model's accuracy and its interpretability. There is outstanding research work combining GNN with the attention mechanism. For instance, GAT [50] and its variants [23, 27, 51, 63] learn to assign attention weights to each of the neighbors of a given node in a graph, based on their relevance to the node's representation. The attention mechanism allows the model to selectively focus on the most important neighbors, rather than treating all neighbors equally. The results from these works are promising. Although attention is utilized during the training process, it is not a straightforward task to employ it directly for the purpose of enabling a GAT network to perform graph sampling.

3 PRELIMINARIES

Formally, let Φ be a graph-based model architecture (e.g., a GCN) that can process graph-structure data, and B be the number of nodes we try to sample. Now given a large graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ where $\mathcal{V}$ and $\mathcal{E}$ are node and edge sets, respectively, and $N = |\mathcal{V}| > B$, then our goal aims to select a subgraph represented as $\mathcal{G}\langle\mathcal{V}^b\rangle = (\mathcal{V}^b, \{uv | uv \in \mathcal{E} \text{ and } u, v \in \mathcal{V}^b\})$:

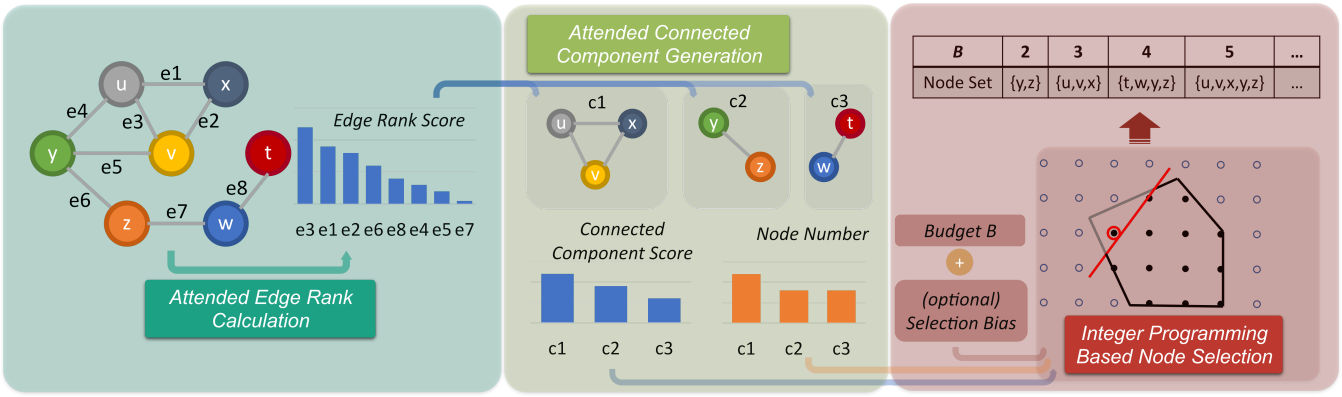

Figure 1: Illustration of the graph sampling process in *GraphSAID*. It is important to note that stages 1 (*Attended Edge Rank Calculation*) and 2 (*Attended Connected Component Generation*) only need to be executed once for the entire graph, after which stage 3 (*Integer Programming Based Node Selection*) can perform graph sampling concurrently for different budget values of B.

$$\mathcal{V}^b = \arg\min_{\mathcal{V}^b} \mathbf{d}_\Phi(\mathcal{G}\langle\mathcal{V}^b\rangle, \mathcal{G}),\ s.t.\ \mathcal{V}^b \subseteq \mathcal{V}\ and\ |\mathcal{V}^b| = B \quad (1)$$

where $\mathbf{d}_\Phi(.,.) \in \mathbb{R}$ quantifies the difference of its two inputs for a given Φ. In particular, $\mathbf{d}_\Phi(\mathcal{G}\langle\mathcal{V}^b\rangle, \mathcal{G})$ can be considered as a measurement of the subgraph effectiveness by calculating the semantic difference between $\mathcal{G}\langle\mathcal{V}^b\rangle$ and its corresponding whole graph $\mathcal{G}$. In addition, to obtain the semantic difference, we need a helper model architecture Φ. It is worth noting that there are several approaches to constructing $\mathbf{d}$ [25] and we focus on the inference performance dissimilarity between a subgraph $\mathcal{G}\langle\mathcal{V}^b\rangle$ and its corresponding full graph $\mathcal{G}$, using the same helper model architecture Φ. Also, note that $\mathbf{d}_\Phi$ is evaluated on the unchanged testing set.

4 OPTIMIZATION VIA EDGE RANKING

When it comes to semantically graph sampling, one might consider using node ranking based methods to rank nodes and then select the high-rank node-set, however, we opt for edge ranking for the following reasons:

(1) When we globally rank edges and subsequently remove some of them, the whole node set can be preserved while the complexity of the graph can be reduced. This is especially important for the cases in which non-trivial inherent edge noise emerges in the whole graph [16].
(2) As the number of removed edges increases, more connected components are developed. This will offer cluster-level computations that exhibit enhanced sampling efficiency.
(3) Edges can serve as a versatile medium for conveying node interaction information, such as attention [49].

To this end, we propose *GraphSAID*, a three-stage method (as shown in Figure 1) to perform the graph sampling process: (1) *the Attended Edge Rank Calculation* process (in Section 4.1), where we define a *total attention* amount, which quantifies the overall significance in facilitating the prediction of other nodes for each edge. (2) the *Attended Connected Component Generation* process (in Section 4.2), where we sort edges by the *total attention amount* and then progressively merge the sorted edges into the attended connected components. (3) the *Integer Programming Based Node Selection* method (in Section 4.3), which employs an Integer Programming approach with adaptable constraints to efficiently and effectively select the generated connected components.

4.1 Attended Edge Rank Calculation

In the first stage, we aim to create ranking scores for each edge. While traditional methods consider *centrality* [11, 19] and its modifications to perform edge ranking [13], we leverage the learned attention, which has successfully shown its ability to significantly improve the performance in various tasks [51, 55, 64]. In particular, our approach embraces the widely acclaimed GCN model *GraphSAGE* [20] that has paved the way for a suite of influential subsequent works, including *FastGCN* [8] and *GraphSAINT* [61]. More specifically, *GraphSAGE* utilizes:

$$h_v^t \leftarrow \sigma(W \cdot \text{MEAN}(\{h_v^{t-1}\} \cup \{h_u^{t-1}, \forall u \in \mathcal{N}(v)\}) \quad (2)$$

to perform feature transformation during the learning phase, where h^t is used to represent a node's representation at the "t" aggregator step, $h^{t=0}$ indicates the original node feature and $\mathcal{N}(v)$ is the node neighbor set of v. We extend Eq. 2 to leverage attention as:

$$\begin{aligned}
&h_v^t \leftarrow \sigma(\text{norm}(W_D^{t-1} \cdot (\{h_Q^{t-1}\} \cup A(h_Q^{t-1}, h_K^{t-1}, h_V^{t-1})))\\
&h_Q^{t-1} \leftarrow W_Q^{t-1} \cdot h_v^{t-1}\\
&h_K^{t-1} \leftarrow (\ W_K^{t-1} \cdot h_{v_1}^{t-1}, W_K^{t-1} \cdot h_{v_2}^{t-1}, ..., W_K^{t-1} \cdot h_{v_s}^{t-1}\ ) \quad (3)\\
&h_V^{t-1} \leftarrow (\ W_V^{t-1} \cdot h_{v_1}^{t-1}, W_V^{t-1} \cdot h_{v_2}^{t-1}, ..., W_V^{t-1} \cdot h_{v_s}^{t-1}\ )\\
&A(Q,K,V) = softmax(QK^T / \sqrt{d_k})\ V
\end{aligned}$$

where $v_1, v_2, ..., v_s \in \mathcal{N}(v)$, "norm" refers to layer normalization with the default settings specified in [2], $W_D^{t-1}, W_Q^{t-1}, W_K^{t-1}, W_V^{t-1}$ are learning parameters. Note that we follow the suggested setting $t = 2$ presented in the original work [20]. Extending Eq. 2 to Eq. 3, we can easily obtain learned attention about local neighbors for each node after we trained the *GraphSAGE* model. Fig. 2 illustrates such learned attention distribution attached to each node. Specifically, each node will have a tree structure attention distribution where the tree depth is equal to $t = 2$ (in Eq. 3). In addition, from Eq. 3 it

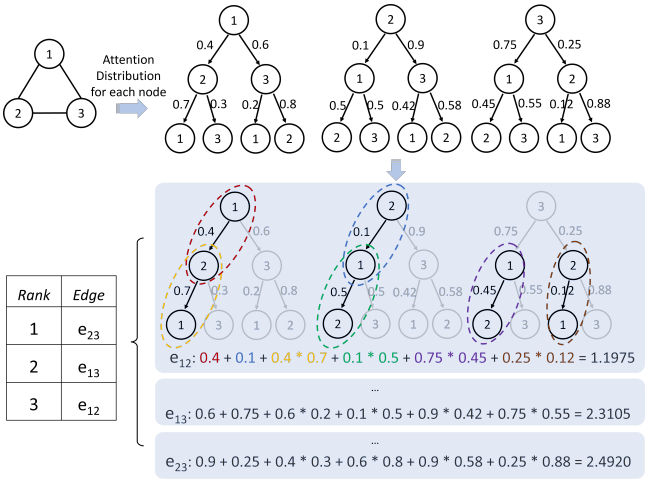

Figure 2: Example of the learned attention distribution for each node (best view in color). Top left: a complete graph with 3 nodes. Top right: attention distribution for each node (presented as a root node in each tree). Each branch is associated with a value showing the attention amount from a parent to its child. Bottom right: edge ranking score calculation. Bottom left: final edge ranking.

can be easily derived that for each parent node of the tree, the sum of the attention values of its first-order children is 1.

Given the learned attention where each node maintains a local tree structure distribution, we need to aggregate and subsequently transform them to associate with each edge so that we can perform edge scoring/ranking for further selection processes. To do that, our strategy is that for each edge we accumulate all the attention that will flow through it across all the attention distribution trees. In particular, if an edge of the graph appears in a branch of a tree, we need to accumulate the attention value associated with that branch. As illustrated in Fig. 2, there are two types of attention according to the depth of the parent node, i.e., from depth 2 to 1 and 1 to 0, respectively. Since the aggregation follows a bottom-up approach [20], when the attention flows from depth 1 to 0, we simply accumulate the associated attention value for the edge. In contrast, when the attention flows from depth 2 to 1, we discount it since this attention value only affects an intermediate node rather than the root node.

To formulate the efficient aggregation illustrated in Fig. 2, we firstly design two sparse matrices $A_1 \in \mathcal{R}^{N\times N}$ and $A_2 \in \mathcal{R}^{N\times N\times N}$ to represent the attention flowed from depth 1 to 0 and 2 to 1, respectively, where N is the total number of nodes of the given graph. The entry a_{ij} of A_1 represents the attention associated with the edge from node i (root) to node j (child at depth 1), while the entry c_{ljk} of A_2 indicates the attention from node j (parent at depth 1) to node k (child at depth 2) in the lth tree. For example, A_1 and A_2 of the Fig. 2 are:

$$A_1 = \begin{pmatrix} 0 & 0.4 & 0.6 \\ 0.1 & 0 & 0.9 \\ 0.75 & 0.25 & 0 \end{pmatrix} \tag{4}$$

$$A_2 = \left(\begin{pmatrix} 0 & 0 & 0 \\ 0.7 & 0 & 0.3 \\ 0.2 & 0.8 & 0 \end{pmatrix}, \begin{pmatrix} 0 & 0.5 & 0.5 \\ 0 & 0 & 0 \\ 0.42 & 0.58 & 0 \end{pmatrix}, \begin{pmatrix} 0 & 0.45 & 0.55 \\ 0.12 & 0 & 0.88 \\ 0 & 0 & 0 \end{pmatrix} \right) \tag{5}$$

In this way, we can use A_1 and A_2 to represent our aggregated edge ranking score matrix:

$$RankScore = A_1 + \sum_{i=1}^{N} (A_1' A_2)[i,:,:] \tag{6}$$

$$A_1' = \left(I \odot A_1[1,:]^T \ 1^T,\ I \odot A_1[2,:]^T \ 1^T, ..., I \odot A_1[N,:]^T \ 1^T \right) \tag{7}$$

where I is the identy matrix, $\odot$ is the element-wise product and $A_1[i,:]$ is the ith row of A_1. Finally, we can obtain the edge rank scores for the edge (u,v) as:

$$e_{uv} = RankScore(u,v) + RankScore(v,u) \tag{8}$$

Using the example of Fig. 2, we can derive that A_1' and $RankScore$:

$$A_1' = \left(\begin{pmatrix} 0 & 0 & 0 \\ 0 & 0.4 & 0 \\ 0 & 0 & 0.6 \end{pmatrix}, \begin{pmatrix} 0.1 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0.9 \end{pmatrix}, \begin{pmatrix} 0.75 & 0 & 0 \\ 0 & 0.25 & 0 \\ 0 & 0 & 0 \end{pmatrix} \right) \tag{9}$$

$$RankScore = \begin{pmatrix} 0 & 0.7875 & 1.0625 \\ 0.41 & 0 & 1.24 \\ 1.248 & 1.252 & 0 \end{pmatrix} \tag{10}$$

In this way, we can facilitate the sampling process by using matrix parallel computation.

4.2 Attended Connected Component Generation

After we obtain edge ranking score e_{uv} for each edge, in the second stage, we broaden our focus from the edge level to the group level, i.e., analysis of connected components. This action offers two advantages. Firstly, it maintains the semantic coherence [40] of local neighbors within a connected component (CC). Secondly, sampling on CCs is more efficient compared to node-by-node sampling.

Given e_{uv} for each edge in $\mathcal{G}$, we arrange them as a descending ordered set $E = (e^1, e^2, ..., e^m)$. Subsequently, leveraging the ordered set E, we employ a greedy approach to select and merge edges in the descending rank order. As edges merge to form connected components, we incorporate a threshold value τ that regulates the upper limit of each connected component's size.

More specifically, we first create an empty collection $\mathcal{L}$ which will contain node sets. Then we take edges from E one by one. Given a taken edge (u,v), there will be four cases for $\mathcal{L}$ to handle:

- If neither u nor v can be found in any node set in $\mathcal{L}$, then we append {u,v} to $\mathcal{L}$
- Given $n \in \{u,v\}$ can be found in a node set $s \in \mathcal{L}$, if $|s| < \tau$, we merge $\{u,v\}$ to s, otherwise, we create $\{u,v\} \setminus n$ to append to $\mathcal{L}$
- If both u,v can be found in two node sets s_1 and s_2 of $\mathcal{L}$ respectively with $|s1| < \tau$ and $|s2| < \tau$, then we merge s_1 and s_2 inside $\mathcal{L}$.

Algorithm 1 Attended Connected Component Generation

Require: $\mathcal{G}, \tau, E$
Ensure: $\mathcal{L}$ (the collection of connected components as node sets)
List $\mathcal{L} \leftarrow [\{\}]$
Queue $Q \leftarrow E$
while $Q \neq null$ **do**
  Edge $(u, v) \leftarrow Q.pop()$
  if $\forall s \in \mathcal{L} : u \notin s \wedge v \notin s$ **then**
    $\mathcal{L}.append(\{u, v\})$
  else if $\exists s \in \mathcal{L}\ s.t.\ u \in s \ \vee\ v \in s$ **then**
    $\begin{cases} \text{add } u \text{ and } v \text{ to } s, & \text{if } |s| < \tau \\ \text{add } \{u, v\} \setminus s \text{ to } \mathcal{L}, & \text{otherwise} \end{cases}$
  else if $\exists s_1, s_2 \in \mathcal{L}\ s.t.\ u \in s_1 \text{ and } v \in s_2$ **then**
    merge s_1 and s_2 *in* $\mathcal{L}$
  end if
end while

This process will go through all edges of E and we show more implementation details in algorithm 1 (for a more comprehensive algorithm, kindly refer to our appendix [33]). At the end of this stage, we will obtain a vector $c = (c_1, c_2, ..., c_l)^T$ where $l = |\mathcal{L}|$, containing the *total attention amount* summed within each connected component (i.e., each node set in $\mathcal{L}$). Specifically, we have:

$$c_i = \sum_{(u,v) \in \mathcal{G}\langle s_i \rangle} e_{uv} \tag{11}$$

where $s_i \in \mathcal{L}$ is the ith connected component stored in $\mathcal{L}$.

4.3 Integer Programming Based Node Selection

In the third (final) stage, we perform connected component selection to achieve graph sampling. Given that attention is crucial for understanding semantic nuances [51, 55], our goal is to select optimal subsets of CCs so that the *total attention amount* is maximized and the summed size of the selected subsets does not exceed B. In other words, we seek:

$$\begin{aligned} &maximize\ c^T x \\ s.t.\ &v^T x \ \leq\ B \end{aligned} \tag{12}$$

where $x \in \{0, 1\}^{|c|}$, $v \in \mathbb{R}^{|c|}$. Here x represents a final selection vector on $\mathcal{L}$, v is the vector containing CC size in $\mathcal{L}$, i.e., $v = (|s_1|, |s_2|, ..., |s_{|c|}|)$, where $s_1, s_2, ..., s_{|c|} \in \mathcal{L}$. Note that Eq. 12 naturally fits the Integer Programming formulation [6], therefore we can apply the standard branch and bound method [28] to solve Eq. 12 to obtain $\mathcal{V}^b$.

4.3.1 Extend Flexible Constraints. Note that we can design or interpret any kind of user requirements as linear constraints in Eq. 12 due to the properties of the mixed integer programming [1]. We have devised two general constraints (denoted as *c1* and *c2*) that are widely applicable in practice, aiming to enhance the flexibility

Figure 3: Illustration showcasing the operation of u for different types of p in E.q 14. Left: skewed p. Right: flat p.

of our node selection process:

$$\begin{aligned} &maximize\ c^T x \\ s.t.\ &v^T x \ \leq\ B \\ &Qx \ \geq\ \epsilon \qquad (c1) \\ &Qx \ \leq\ u \qquad (c2) \end{aligned} \tag{13}$$

where $Q \in \mathbb{N}^{m \times |c|}$, $\epsilon \in \mathbb{N}^m$, $u \in \mathbb{N}^m$, m is the number of different labels and $|c|$ is the number of CCs. Q is the matrix storing the label counts in each node set of $\mathcal{L}$. For example, if $\mathcal{L} = \{\{a, b, c\}, \{d, e\}\}$ and the label of a, b, c, d, e are $1, 1, 1, 0, 1$ respectively, then $Q = \begin{pmatrix} 3 & 0 \\ 1 & 1 \end{pmatrix}$. In addition, ϵ is the "minimum number" constraint vector to ensure the existence of minor-class data (e.g., in the Cora dataset [35]). Likewise, u represents the maximum limit for each sampled class number. More specifically, we set

$$\begin{aligned} \epsilon &= \mathbf{1} \ \cdot\ \min\{\gamma B, \eta\} \\ u &= B \cdot (\alpha(p - \frac{1}{2}) + p + \delta) \end{aligned} \tag{14}$$

where γ represents the minimum required proportion of B that must be present in each class. η is the quantity required for each class. Therefore, the control of ϵ can be determined by the user through either a proportion value or an absolute value. For the maximum limit u, p is a vector of presence ratio for each class (e.g., $p = (0.2, 0.8)$ represents a two-class situation where the probability of class 1 and 2 are 0.2 and 0.8 respectively). When we consider whether to take a high-attention CC containing the minority class, we relax the restriction by raising the maximum limit. This is because a high-attention CC containing the minority class is crucial and more valuable for the minority semantic aspect of the graph, particularly when the class distribution is imbalanced [36, 39]. Therefore, we adjust p by $\alpha(p - \frac{1}{2})$ where $0 < \alpha < 1$ to appropriately make the minority class associated with a higher maximum limit. δ is a small globally increasing value when p is relatively flatten. Fig. 3 illustrates the functionality of u. By incorporating these constraints, the sampling process becomes more flexible and aligns with the specific needs of downstream learning tasks [47, 58].

5 EXPERIMENTS

In this section, we conduct extensive experiments on four datasets to verify the effectiveness of *GraphSAID*. Specifically, we assess the learning performance of various sampling outcomes to gain insights into which results better preserve semantic integrity. Additionally, we demonstrate through case studies that *GraphSAID* has the ability to easily leverage selection bias to enhance subgraph selection performance.

		Elliptic				Cora			Citeseer			GPRG		
B	Method	R_{attn}	Acc	F1	Loss	R_{attn}	Acc	Loss	R_{attn}	Acc	Loss	R_{attn}	Acc	Loss
20%	FF	0.053	0.687	0.192	0.598	0.072	0.477	2.465	0.197	0.259	2.931	0.097	0.766	0.666
20%	SB	0.053	0.869	0.350	0.434	0.071	0.623	1.264	0.197	0.423	1.862	0.066	0.841	0.463
20%	SRW	0.046	0.892	0.411	0.421	0.072	0.671	1.096	0.196	0.364	2.355	0.055	0.847	0.468
20%	RWF	0.045	0.841	0.338	0.465	0.072	0.612	1.258	0.192	0.430	1.774	0.067	0.812	0.536
20%	ORC-sub	oom	oom	oom	oom	0.080	0.494	2.301	0.204	0.380	1.946	0.115	0.778	0.610
20%	Ours	**0.225**	0.943	**0.479**	**0.369**	**0.139**	0.558	1.742	**0.218**	0.578	1.231	**0.200**	**0.862**	**0.432**
20%	*Ours*†	**0.225**	0.942	0.473	0.370	0.121	**0.716**	**0.937**	**0.218**	0.576	1.247	**0.200**	**0.862**	**0.432**
20%	*Ours*‡	**0.225**	**0.943**	0.478	**0.369**	0.121	**0.716**	**0.937**	**0.218**	**0.604**	**1.155**	**0.200**	**0.862**	**0.432**
40%	FF	0.107	0.881	0.398	0.432	0.157	0.690	1.092	0.406	0.474	1.482	0.256	0.819	0.588
40%	SB	0.106	0.919	0.428	0.396	0.159	0.759	0.782	0.416	0.523	1.430	0.219	0.828	0.471
40%	SRW	0.092	0.910	0.398	0.407	0.160	0.752	0.781	0.418	0.522	1.410	0.195	0.860	0.452
40%	RWF	0.045	0.841	0.338	0.465	0.157	0.725	0.940	0.415	0.531	1.362	0.192	0.861	0.451
40%	ORC-sub	oom	oom	oom	oom	0.171	0.638	1.365	0.420	0.543	1.355	0.295	0.860	**0.421**
40%	Ours	**0.379**	**0.943**	0.505	**0.369**	**0.262**	0.738	0.914	0.432	0.618	1.175	**0.404**	**0.862**	0.427
40%	*Ours*†	**0.379**	0.942	0.501	0.370	0.260	**0.770**	0.764	0.432	**0.623**	**1.143**	**0.404**	**0.862**	0.427
40%	*Ours*‡	**0.379**	**0.943**	**0.506**	**0.369**	0.260	**0.770**	**0.753**	**0.434**	0.618	1.146	**0.404**	**0.862**	0.427

Table 1: Learning performance comparison using different sampled subgraphs on four datasets. The best performance is in bold. "oom" means out-of-memory issue during the computation.

	GRPG	Cora	Citeseer	Elliptic
AvgCC	0.0286	0.2407	0.1447	0.01376
Density	0.0137	0.0014	0.0007	1.1288e-5
#C	2	7	6	2
$\lvert\mathcal{V}\rvert$	1998	2708	3312	203,769
$\lvert\mathcal{E}\rvert$	13,692	5429	4715	234,355
#Dim	3	1433	3703	165
#W_1	3×3	1433×64	3703×64	3703×64
#W_2	3×2	64×7	64×6	100×2

Table 2: Statistics of the datasets and the settings of Φ. *AvgCC* and *Density* are the average clustering coefficient and graph density. #C is the total number of classes, while $\lvert\mathcal{V}\rvert$ and $\lvert\mathcal{E}\rvert$ are the numbers of nodes and edges, respectively. #Dim represents the number of dimension of node feature. #W_1 and #W_2 is the number of neurons in the two GCN layers in Φ.

5.1 Datasets and Experimental Settings

We use one synthetic graph GRPG (Gaussian Random Partition Graph) and three labeled real graphs, including Citeseer [44], Cora [35], and Elliptic [53] for our experiment. The four datasets are used for node classification, where GRPG and Elliptic are for binary classification, and Cora and Citeseer are for multi-class classification. We implement Φ with two GCN layers for all datasets, while in each dataset different numbers of neurons will be applied. Table 2 summarized the dataset and experiment settings. In addition, we heuristically choose $\tau = 50$, $\alpha = 0.25$ and $\delta = 0.05$. All experiments are conducted on a machine with 96 CPU cores, 377G RAM, and 8 NVIDIA-1080 graphic cards. For convenience and clarity, when B is associated with a fraction number, it indicates the proportion of the total training nodes in a particular dataset (e.g., see Table 1).

5.2 Baselines

We define the baselines using: (1) Simple Random Walks (SRW) [41]: This method serves as the foundation for a wide range of graph sampling methods that rely on exploration strategies. (2) Random Walk sampling with Fly-back (RWF) [48]: This method has demonstrated its success in analyzing the bitcoin network. (3) Snow Ball sampling (SB) [18]: a restricted version of bread first search. (4) Forest Fire sampling (FF) [31]: a parameterized stochastic version of SB. (5) Ollivier Ricci curvature Gradient-based subsampling (ORG-sub) [56]: This technique utilizes a greedy exploration approach based on Ricci curvature [43]. Additionally, our work is denoted in three versions: (1) *Ours*: This version removes both (*C1*) and (*C2*) in Eq. 13. (2) *Ours*†: This version removes (*C1*) in Eq. 13. (3) *Ours*‡: This version preserves the complete form of Eq. 13.

5.3 Experimental Results and Analysis

5.3.1 Overall Performance. Table 1 shows the performance of the node classification tasks on four datasets with training data sizes at 20% and 40% of the original (training data are subgraphs containing 20% and 40% number of nodes in the original graphs). We also present 60% and 80% in the appendix [33]. For a given B and dataset, while *GraphSAID* is a deterministic method, for other methods, we conducted 5 experiments each with different seed initialization for statistical reliability. Furthermore, when analyzing the Elliptic dataset, we assess the F1 score specifically for the "minority" class, which refers to illicit nodes. This evaluation is conducted on subsets comprising 20%, 40% of licit and illicit nodes, with unknown nodes excluded [53]. Our results reveal that *GraphSAID* exhibits superior performance when compared to others, especially when B is small. For example, when $B \longrightarrow 20\%$, there are +6.8% F1 score boost in Elliptic dataset, and +5.1%, +4.5%, +17.4% and +1.5% acc boost in Elliptic, Cora, Citeseer and GRPG, respectively. We additionally show the attention retention rate (the rate of attention contained in the subgraph) denoted as R_{attn} for each method. Our empirical results show the success of collecting high-attention CCs for different B across different datasets. Moreover, although most of the time, the difference among *Ours*, *Ours*† and *Ours*‡ is trivial, for the case of

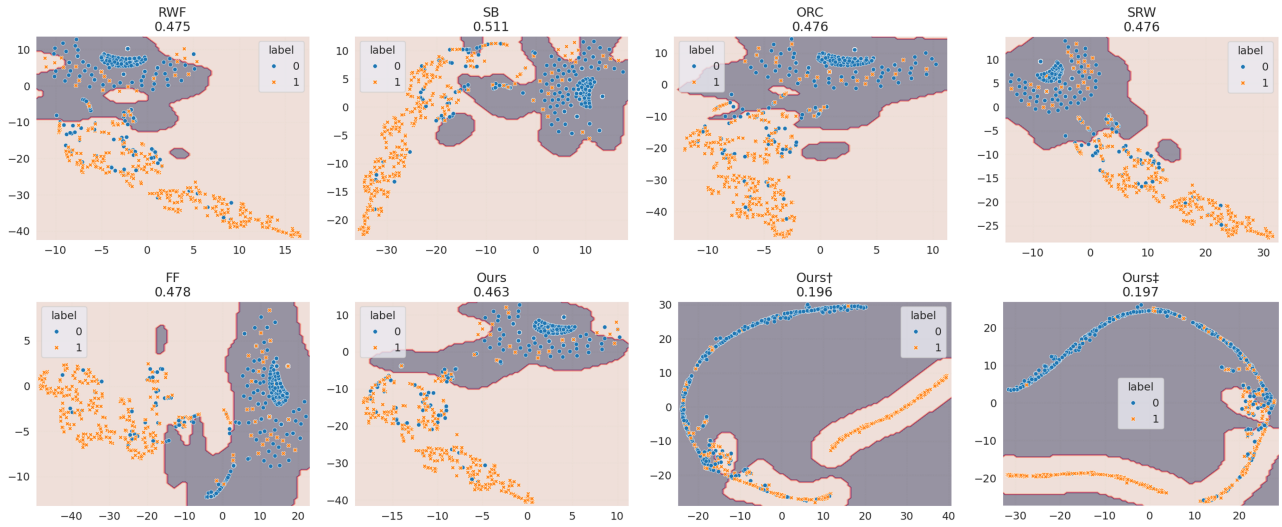

Figure 4: *T-SNE* visualization on GRPG dataset when $B = 20\%$. The number on top of each sub-figure is the KL-divergence (representing projection error) output from the *T-SNE* algorithm. The bright and dark area (outlined by a red border) of each sub-figure is the learned *SVM* decision boundary. Best viewed in color.

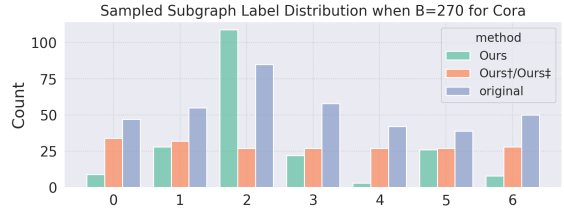

Figure 5: Comparison of sampled subgraph label distribution. The distribution is associated with the experiment result where $B \longrightarrow 20\%$ on the Cora dataset.

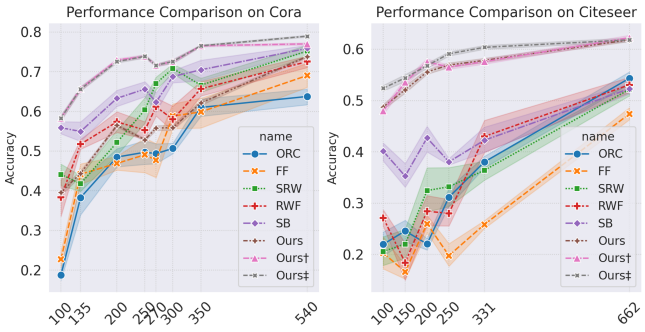

Figure 6: Accuracy for various methods across different small B values in Cora (left) and Citeseer (right) datasets. Note that $B = 270$ (left) and $B = 331$ (right) represent "20%" in Table 1 respectively.

"20% - Cora", both *Ours*† and *Ours*‡ outperform *Ours* by 16% acc enhancement. This is due to the influence of the imbalanced class distribution in the Cora dataset, which can impact the sampling process by causing high-attention connected components (CCs) with a large majority class to overshadow those with minority classes. Fig. 5 shows how using lower or upper limits (in Eq. 14) can affect the sampled subgraph label distribution. We can see that *Ours*† or *Ours*‡ demonstrates a smoother sampled label distribution than that from *Ours*, while the total attention retain rates are similar: 0.139 vs 0.121 (in Table 1).

On the other hand, Table 1 also reveals that when $B \longrightarrow 40\%$, our performance boost is not as significant as when $B \longrightarrow 20\%$. One reason could be that the model performance starts to converge when $B \longrightarrow 40\%$. Since the model performance gain is not linear to the amount of training data. As a result, to investigate more detailed performance differences between our methods and the others, we conduct a more comprehensive comparison by analyzing a lower and denser set of B intervals.

5.3.2 Experiments with Extremely Low Value of B. In this subsection, we explore the performance of various methods under the condition of extremely low values of B. This investigation aims to determine the level at which a small subgraph can maintain its semantic content. Fig. 6 shows that *GraphSAID* successfully utilizes an extremely small subgraph to capture the entire graph's semantics. More importantly, our methods consistently outperform the others, which is important since choosing different methods for different B values is tedious and can be expensive. Additionally, the investigation uncovers the presence of small subgraphs within Cora and Citeseer that encode the representative knowledge of the overall graph. Moreover, *GraphSAID* demonstrates the fastest rates of model learning performance convergence, which is particularly valuable in scenarios where the learning period is restricted, and it can additionally support the development of other models designed for rapid convergence [3].

5.3.3 Visualization on the Learned Features. In order to gain a deeper comprehension of the features learned from small graphs obtained through various sampling methods, we utilize visualization tools, such as *t-SNE*, to examine the embedding patterns within the GRPG dataset. Fig 4 depicts the distribution of learned data points in the GRPG testing set, employing different sampling techniques. To assess the degree of separation between *t-SNE* data

points belonging to different classes, we additionally present an SVM-decision boundary learned from the *t-SNE* data points for each subfigure. By observing these visualizations, we can investigate whether the data point distributions for the two classes are clearly distinguishable. Our result indicates both *Ours* and the other baseline methods exhibit a sparse data point distribution. However, the learned decision boundary in *Ours* only forms a single region for one class. This observation can be interpreted as a positive indication since the GRPG dataset is relatively simple and an expected decision boundary should be "straightforward" and avoid separating the embedding space into multiple segments, given the data point distribution in the embedding space is sparse.

On the other hand, $Ours^{\dagger}$ and $Ours^{\ddagger}$ exhibit distinct, denser, and more clearly defined distribution patterns in the embedding space. Furthermore, the embeddings for both classes demonstrate consistency and homogeneity, indicating successful learning. Additionally, the *t-SNE* projection error (*KL-divergence*) is minimal for $Ours^{\dagger}$ and $Ours^{\ddagger}$, enhancing the reliability of the visualization.

5.3.4 Empirical Run Time Analysis . Besides effectiveness, the selection of sampling methods also hinges on their efficiency, making it a critical factor to consider. Hence, we undertake comprehensive run-time analysis on various methods in this regard. Table 3 shows the results of run time to sample a subgraph by different methods across different B values in the Elliptic dataset, which contains more than 200,000 nodes and edges encoding transaction information. Specifically, while other methods exhibit an upward trend in runtime as B ranges from 20% to 80%, our method *GraphSAID*, conversely maintains a relatively consistent runtime. This is attributed to our utilization of sampling on attended CCs instead of individual nodes. This effectively reduces the size of the candidate set and further decreases runtime. This phenomenon presents both advantages and disadvantages. As depicted in Fig 3, *Ours* exhibits longer runtime compared to other methods for small B values (e.g., 20%). This is because we consider attended CCs from a global perspective even when B is small, whereas other methods randomly select a single node as a starting point and explore locally. However, as B increases, the runtime of *Ours* remains relatively stable, while the runtime of other methods noticeably increases (e.g., SRW and RWF). The reason for this is that as the number of nodes increases, the time taken for each node to perform operations, such as randomly selecting the next neighbor node to explore, will be multiplied.

5.3.5 User Flexibility Control . In section 4.3.1, we discussed the possibility of incorporating additional constraints to accurately capture user requirements in various application scenarios. To illustrate this, let's consider the GRGP dataset as an example. In this case, one option could be to sample a smaller (e.g., $B = 20\%$) semantic subgraph while minimizing the inclusion of unknown labels. This feature is beneficial when a user intends to explore the correlation among the labeled nodes. Fig. 7 depicts the user control process, wherein the ϵ constraint in Eq. 14 is manipulated to achieve desired outcomes. Importantly, both graphs demonstrate similar testing accuracy, with values of 0.830 and 0.844 respectively, indicating that they convey equivalent semantic information, even though their visual representations differ due to distinct graph portfolios.

Method	20%	40%	60%	80%
FF	0.383	0.732	1.321	1.637
SB	2.029	2.551	3.136	3.472
SRW	2.974	5.200	9.599	16.887
RWF	3.746	7.227	13.215	22.314
ORG-sub	-	-	-	-
Ours	4.348	4.344	4.453	4.393
FF	0.004	0.008	0.038	0.139
SB	0.029	0.034	0.039	0.052
SRW	0.022	0.034	0.069	0.277
RWF	0.052	0.122	0.328	0.898
ORG-sub	0.04	4.04	0.096	0.199
Ours	0.393	0.211	0.212	0.375

Table 3: Run time in sampling subgraphs in Elliptic (top) and Citeseer (bottom) using different B values. The ORG-sub is currently experiencing an out-of-memory issue on Elliptic, resulting in the absence of any captured run time.

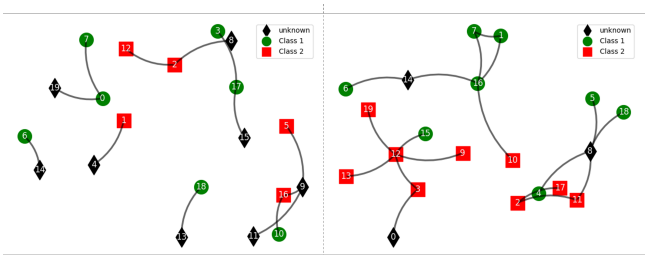

Figure 7: Example of flexible user control on GRPG dataset with $B = 20\%$. Left: $\epsilon = (2, 2)$ for classes 1 and 2. Right: $\epsilon = (7, 7)$. The testing accuracies by learning the two graphs are 0.830 and 0.844 respectively.

6 CONCLUSION

The objective of this work is to sample a subgraph using a new optimization approach that incorporates learned attention within a given budget. To address the previously unexplored challenge of integrating attention into graph sampling problems, we propose a novel three-stage method called *GraphSAID*. *GraphSAID* creates attended connected components from the raw learned attention and then formulate the graph sampling problem as an Integer Programming problem. Our experimental results demonstrate the significant improvements achieved by *GraphSAID*, which also allows for flexible user control to enhance subgraph selection. While *GraphSAID* exhibits advantageous performance, it does require additional computation costs and relies on training a GraphSAGE model to extract attention. Therefore, a promising avenue for future research would be to further reduce computation costs while maintaining effective sampling performance.

ACKNOWLEDGMENTS

We would like to thank our team leader in FCC Analytics, Mr. Ming Lau, for his invaluable guidance and support throughout the entire research process. We are also grateful to the participants from FCC Analytics, especially Mr. Luke Ng and Mr. Wallace Chow, who generously dedicated their time and shared their experiences. Without their cooperation, this study would not have been possible.

REFERENCES

[1] Tobias Achterberg and Roland Wunderling. 2013. Mixed integer programming: Analyzing 12 years of progress. In *Facets of combinatorial optimization: Festschrift for martin grötschel*. Springer, 449–481.

[2] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E Hinton. 2016. Layer normalization. *arXiv*.

[3] Thomas Bachlechner, Bodhisattwa Prasad Majumder, Henry Mao, Gary Cottrell, and Julian McAuley. 2021. Rezero is all you need: Fast convergence at large depth. In *Uncertainty in Artificial Intelligence*. PMLR, 1352–1361.

[4] Yoshua Bengio. 2012. Practical recommendations for gradient-based training of deep architectures. *Neural Networks* 1, 1, 437–478.

[5] Joan Bruna, Wojciech Zaremba, Arthur Szlam, and Yann LeCun. 2013. Spectral networks and locally connected networks on graphs. *arXiv*.

[6] Der-San Chen, Robert G Batson, and Yu Dang. 2011. *Applied integer programming: modeling and solution*. John Wiley & Sons.

[7] Fenxiao Chen, Yun-Cheng Wang, Bin Wang, and C-C Jay Kuo. 2020. Graph representation learning: a survey. *APSIPA TSI* 9, e15.

[8] Jie Chen, Tengfei Ma, and Cao Xiao. 2018. Fastgcn: fast learning with graph convolutional networks via importance sampling. *ICLR*.

[9] Wei-Lin Chiang, Xuanqing Liu, Si Si, Yang Li, Samy Bengio, and Cho-Jui Hsieh. 2019. Cluster-gcn: An efficient algorithm for training deep and large graph convolutional networks. In *SIGKDD*.

[10] Siddhartha Chib and Edward Greenberg. 1995. Understanding the metropolis-hastings algorithm. *The american statistician*.

[11] Kousik Das, Sovan Samanta, and Madhumangal Pal. 2018. Study on centrality measures in social networks: a survey. *SNAM* 8, 1–11.

[12] Michaël Defferrard, Xavier Bresson, and Pierre Vandergheynst. 2016. Convolutional neural networks on graphs with fast localized spectral filtering. *NIPS*.

[13] Ljiljana Despalatović, Tanja Vojković, and Damir Vukicević. 2014. Community structure in networks: Girvan-Newman algorithm improvement. In *MIPRO*.

[14] Christian Doerr and Norbert Blenn. 2013. Metric convergence in social network sampling. In *ACM workshop on HotPlanet*. 45–50.

[15] Pantelis Elinas. 2022. Scalable graph representation learning with Graph Neural Networks | Medium. https://medium.com/@pantelis.elinas/scalable-graph-representation-learning-with-graph-neural-networks-a2ab67e06f9.

[16] James Fox and Sivasankaran Rajamanickam. 2019. How robust are graph neural networks to structural noise? *arXiv*.

[17] Minas Gjoka, Maciej Kurant, Carter T Butts, and Athina Markopoulou. 2010. Walking in facebook: A case study of unbiased sampling of osns. In *IEEE Infocom*.

[18] Leo A Goodman. 1961. Snowball sampling. *The annals of mathematical statistics*.

[19] Felipe Grando, Lisandro Z Granville, and Luis C Lamb. 2018. Machine learning in network centrality measures: Tutorial and outlook. *ACM Computing Surveys (CSUR)* 51, 5, 1–32.

[20] Will Hamilton, Zhitao Ying, and Jure Leskovec. 2017. Inductive representation learning on large graphs. *NIPS* 30.

[21] William L Hamilton. 2020. *Graph representation learning*. Morgan & Claypool Publishers.

[22] David K Hammond, Pierre Vandergheynst, and Rémi Gribonval. 2011. Wavelets on graphs via spectral graph theory. *ACHA*.

[23] Bing Han, Xinyun Zhang, and Shuang Ren. 2022. PU-GACNet: Graph attention convolution network for point cloud upsampling. *Image and Vision Computing*.

[24] Long Jin, Yang Chen, Pan Hui, Cong Ding, Tianyi Wang, Athanasios V. Vasilakos, Beixing Deng, and Xing Li. 2011. Albatross Sampling: Robust and Effective Hybrid Vertex Sampling for Social Graphs. In *ACM MobiArch*.

[25] Mohsen Joneidi, Saeed Vahidian, Ashkan Esmaeili, Weijia Wang, Nazanin Rahnavard, Bill Lin, and Mubarak Shah. 2020. Select to better learn: Fast and accurate deep learning using data selection from nonlinear manifolds. In *CVPR*.

[26] Thomas N Kipf and Max Welling. 2016. Semi-supervised classification with graph convolutional networks. *arXiv*.

[27] Xiangyuan Kong, Weiwei Xing, Xiang Wei, Peng Bao, Jian Zhang, and Wei Lu. 2020. STGAT: Spatial-temporal graph attention networks for traffic flow forecasting. *IEEE Access* 8, 134363–134372.

[28] Arthur H Land and Alston G Doig. 1960. An automatic method of solving discrete programming problems. *EJES*.

[29] Yun-Jung Lee, Su-Do Kim, Jang-Pyo Hong, Hwan-Gue Cho, and Seong-Min Yoon. 2016. Industrial Network Analysis Using Inter-Firm Transaction Data. *IJST*.

[30] Jure Leskovec and Christos Faloutsos. 2006. Sampling from large graphs. In *SIGKDD*. 631–636.

[31] Jure Leskovec, Jon Kleinberg, and Christos Faloutsos. 2005. Graphs over Time: Densification Laws, Shrinking Diameters and Possible Explanations. In *SIGKDD*.

[32] Yujia Li, Oriol Vinyals, Chris Dyer, Razvan Pascanu, and Peter Battaglia. 2018. Learning deep generative models of graphs. *arXiv*.

[33] Ziqi Liu and Laurence Liu. 2024. Appendix of GraphSAID: Graph Sampling via Attention based Integer Programming Method. In *https://github.com/finalyXG/GraphSAID/blob/main/AAMAS2024_supplemental_420.pdf*.

[34] Arun S Maiya and Tanya Y Berger-Wolf. 2010. Sampling community structure. In *Proceedings of the 19th international conference on World wide web*. 701–710.

[35] Andrew Kachites McCallum, Kamal Nigam, Jason Rennie, and Kristie Seymore. 2000. Automating the construction of internet portals with machine learning. *Information Retrieval*.

[36] Maria Carolina Monard and GEAPA Batista. 2002. Learning with skewed class distributions. *Advances in Logic, Artificial Intelligence and Robotics* 85, 173–180.

[37] Nagarajan Natarajan, Inderjit S Dhillon, Pradeep K Ravikumar, and Ambuj Tewari. 2013. Learning with noisy labels. *NIPS* 1, 1.

[38] Chien-Chun Ni, Yu-Yao Lin, Feng Luo, and Jie Gao. 2019. Community detection on networks with ricci flow. *Scientific reports* 9, 1, 1–12.

[39] Ronaldo C Prati, Gustavo EAPA Batista, and Maria Carolina Monard. 2004. Learning with class skews and small disjuncts. In *AAI*. Springer.

[40] Xiaojuan Qi, Renjie Liao, Jiaya Jia, Sanja Fidler, and Raquel Urtasun. 2017. 3d graph neural networks for rgbd semantic segmentation. In *ICCV*. 5199–5208.

[41] Bruno Ribeiro and Don Towsley. 2010. Estimating and sampling graphs with multidimensional random walks. In *SIGCOMM*.

[42] Bruno Ribeiro and Don Towsley. 2012. On the estimation accuracy of degree distributions from graph sampling. In *CDC*.

[43] Gregorio Ricci-Curbastro. [n.d.]. Ricci curvature - Wikipedia. https://en.wikipedia.org/wiki/Ricci_curvature.

[44] Ryan A. Rossi and Nesreen K. Ahmed. 2015. The Network Data Repository with Interactive Graph Analytics and Visualization. In *AAAI*.

[45] Benedek Rozemberczki, Oliver Kiss, and Rik Sarkar. 2020. Little ball of fur: a python library for graph sampling. In *CIKM*. 3133–3140.

[46] Franco Scarselli, Marco Gori, Ah Chung Tsoi, Markus Hagenbuchner, and Gabriele Monfardini. 2008. The graph neural network model. *TNN*.

[47] Mingxing Tan and Quoc Le. 2019. Efficientnet: Rethinking model scaling for convolutional neural networks. In *ICML*. Long Beach, California, 6105–6114.

[48] Bishenghui Tao, Hong-Ning Dai, Jiajing Wu, Ivan Wang-Hei Ho, Zibin Zheng, and Chak Fong Cheang. 2021. Complex network analysis of the bitcoin transaction network. *TCAS-II*.

[49] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. *NIPS* 30.

[50] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, and Yoshua Bengio. 2017. Graph attention networks. *arXiv*.

[51] Xiao Wang, Houye Ji, Chuan Shi, Bai Wang, Yanfang Ye, Peng Cui, and Philip S Yu. 2019. Heterogeneous graph attention network. In *WWW*. 2022–2032.

[52] Watts and Strogatz. [n.d.]. Clustering coefficient - Wikipedia. In *https://en.wikipedia.org/wiki/Clustering_coefficient*.

[53] Mark Weber, Giacomo Domeniconi, Jie Chen, Daniel Karl I Weidele, Claudio Bellei, Tom Robinson, and Charles E Leiserson. 2019. Anti-money laundering in bitcoin: Experimenting with graph convolutional networks for financial forensics. *arXiv*.

[54] Felix Wu, Amauri Souza, Tianyi Zhang, Christopher Fifty, Tao Yu, and Kilian Weinberger. 2019. Simplifying graph convolutional networks. In *ICML*. PMLR.

[55] Haiyan Wu, Zhiqiang Zhang, Shaoyun Shi, Qingfeng Wu, and Haiyu Song. 2022. Phrase dependency relational graph attention network for Aspect-based Sentiment Analysis. *Knowledge-Based Systems* 236, 107736.

[56] Shushan Wu, Huimin Cheng, Jiazhang Cai, Ping Ma, and Wenxuan Zhong. 2022. Subsampling in Large Graphs Using Ricci Curvature. In *The Eleventh International Conference on Learning Representations*.

[57] Zonghan Wu, Shirui Pan, Fengwen Chen, Guodong Long, Chengqi Zhang, and S Yu Philip. 2020. A comprehensive survey on graph neural networks. *IEEE transactions on neural networks and learning systems* 32, 1, 4–24.

[58] Cheng Yang, Jiawei Liu, and Chuan Shi. 2021. Extract the knowledge of graph neural networks and go beyond it: An effective knowledge distillation framework. In *Proceedings of the web conference 2021*. 1227–1237.

[59] Rex Ying, Jiaxuan You, Christopher Morris, Xiang Ren, William Hamilton, and Jure Leskovec. 2018. Attention-Based Graph Neural Network for Semi-Supervised Learning. In *NIPS*.

[60] Hao Yuan, Haiyang Yu, Shurui Gui, and Shuiwang Ji. 2022. Explainability in graph neural networks: A taxonomic survey. *IEEE transactions on pattern analysis and machine intelligence* 45, 5, 5782–5799.

[61] Hanqing Zeng, Hongkuan Zhou, Ajitesh Srivastava, Rajgopal Kannan, and Viktor Prasanna. 2019. Graphsaint: Graph sampling based inductive learning method. *arXiv*.

[62] Jiaxu Zhang, Hongxu Chen, Xi Chen, Jun Liu, Wei Wang, and Zhenhong Wu. 2020. Relation-aware Graph Convolutional Networks with Attention Mechanism for Traffic Flow Forecasting. In *CIKM*. Virtual Event, Ireland.

[63] Yiding Zhang, Xiao Wang, Chuan Shi, Xunqiang Jiang, and Yanfang Ye. 2021. Hyperbolic graph attention network. *IEEE Trans. Big Data*.

[64] Wenbo Zheng, Lan Yan, Chao Gou, Zhi-Cheng Zhang, Jun Jason Zhang, Ming Hu, and Fei-Yue Wang. 2021. Pay attention to doctor–patient dialogues: multimodal knowledge graph attention image-text embedding for COVID-19 diagnosis. *Information Fusion* 75, 168–185.

[65] Jie Zhou, Ganqu Cui, Shengding Hu, Zhengyan Zhang, Cheng Yang, Zhiyuan Liu, Lifeng Wang, Changcheng Li, and Maosong Sun. 2020. Graph neural networks: A review of methods and applications. *AI open* 1, 57–81.