SpanGNN: Towards Memory-Efficient Graph Neural Networks via Spanning Subgraph Training

Xizhi Gu[1], Hongzheng Li[1], Shihong Gao[2], Xinyan Zhang[1], Lei Chen[2], and Yingxia Shao[1(✉)]

[1] Beijing University of Posts and Telecommunications, Beijing , China
{guxizhi,Ethan_Lee,xianxx492,shaoyx}@bupt.edu.cn
[2] Hong Kong University of Science and Technology, Guangzhou, China
sgaoar@connect.ust.hk, leichen@cse.ust.hk

Abstract. Graph Neural Networks (GNNs) have superior capability in learning graph data. Full-graph GNN training generally has high accuracy, however, it suffers from large peak memory usage and encounters the Out-of-Memory problem when handling large graphs. To address this memory problem, a popular solution is mini-batch GNN training. However, mini-batch GNN training increases the training variance and sacrifices the model accuracy. In this paper, we propose a new memory-efficient GNN training method using spanning subgraph, called SpanGNN. SpanGNN trains GNN models over a sequence of spanning subgraphs, which are constructed from empty structure. To overcome the excessive peak memory consumption problem, SpanGNN selects a set of edges from the original graph to incrementally update the spanning subgraph between every epoch. To ensure the model accuracy, we introduce two types of edge sampling strategies (i.e., variance-reduced and noise-reduced), and help SpanGNN select high-quality edges for the GNN learning. We conduct experiments with SpanGNN on widely used datasets, demonstrating SpanGNN's advantages in the model performance and low peak memory usage.

1 Introduction

Graph Neural Networks (GNNs) achieve the state-of-the-art performance on graph learning tasks, such as node classification [16,19,27], link prediction [23,39] and graph classification [9,37]. They have been widely used in various domains, like social network analysis [10,26], recommendation [17,34,35], healthcare [1,7], short-term load forecasting [25] and bio-informatics [11,21]. Most of GNNs [16, 19,27,36] follow the message passing paradigm [13], which exploits graph topology and node/edge features simultaneously. In this paradigm, the edge related memory consumption predominantly influences the amount of peak GPU memory [30]. The edge calculation of GNNs involves four operations, which are collection, messaging, aggregation, and feature updating. The four operations require

A. Bifet et al. (Eds.): ECML PKDD 2024, LNAI 14943, pp. 250–266, 2024.
https://doi.org/10.1007/978-3-031-70352-2_15

the storage of intermediate results (e.g., the updated embedding of edge feature and the aggregation of feature embedding), which are used for the gradient calculation in the subsequent backward propagation process. According to the existing empirical studies [30], the peak memory consumption can be up to 100 times of the size of dataset itself. As a result, the high memory usage of the edge calculation restricts the GNNs scaling to large graphs.

Since the edge calculation is the main factor of high memory usage, an intuitive idea is to reduce the number of edges for training. Sampling is a standard technique to generate graphs with few edges. It has been well studied in the mini-batch training. Many works [6,8,16,37,38] use various sampling techniques to create mini-batches, which are subgraphs rooted from a limited number of target nodes. Although mini-batch training is scalable and memory-efficient, it brings in non-negligible training variance and heavily compromises model accuracy. Full-graph GNN training is more accurate than mini-batch training [18]. However, the existing complex sampling methods cannot be efficiently adopted to the full-graph GNN training. The sampling step is time-consuming and becomes the efficiency bottleneck for GNN training on large graphs [30]. Unlike the sampling technique, DropEdge [22] randomly drops edges of the original graph during the full-graph training. It not only reduces the size of peak memory, but also is scalable to large graphs. Nonetheless, DropEdge also suffers from a prominent model accuracy loss as the edge drop ratio increases, especially on large graphs. This limitation arises because DropEdge treats all edges equally and ignores the inherent structure of the original graph. Therefore, *how to develop a memory-efficient and accurate full-graph GNN learning method remains unsolved*.

In this paper, we propose SpanGNN to achieve memory-efficient full-graph GNN training while guaranteeing the model accuracy. First, SpanGNN trains GNN models across a sequence of spanning subgraphs, which are constructed from empty structure. Each spanning subgraph contains significantly fewer edges than those present in the original graphs, thus effectively reducing the peak memory footprint. Furthermore, in each training epoch, SpanGNN selects a set of edges from the original graph to incrementally update the spanning subgraph that used in the previous epoch. Meanwhile, the updated spanning graph always satisfies the sparsity constraint defined by the edge ratio α (See the definition in Sect. 2.2). Second, to guarantee the model accuracy and training efficiency, we propose a fast quality-aware edge selection method for SpanGNN. We analyze the training variance and gradient noise that inherent in the spanning subgraph training framework, and propose variance-reduced sampling and gradient-noise reduced sampling strategies, respectively, to help SpanGNN selects a set of high-quality edges and guarantee the model accuracy. However, it is expensive to directly apply the above two sampling strategies over large graphs, we introduce a two-step sampling method to speed up the edge selection process. Extensive experiments demonstrate that SpanGNN is capable of saving over 40% of GPU memory usage without compromising training performance.

Our main contributions are summarized as follows:

- We propose SpanGNN that supports memory-efficient and accurate full-graph GNN training on large graphs. The new method reduces the peak memory usage significantly during the training, meanwhile achieving high model accuracy.
- We introduce a fast quality-aware edge selection method to alleviate the negative impacts caused by spanning subgraph training and ensure training efficiency.
- We analyze the connection between SpanGNN and curriculum learning [3]. With the help of quality-aware edge selection, SpanGNN selects edges that are highly beneficial to the learning in priority, and then gradually uses edges with low benefits.
- Experimental results on widely used datasets demonstrate that SpanGNN reduces peak memory usage effectively while guaranteeing that the model accuracy is almost equivalent to the one of full graph training.

2 Preliminary

2.1 Graph Neural Networks

The general matrix formulation of GNN models is as follows:

$$Z^{(l)} = PH^{(l-1)}W^{(l-1)}, \tag{1}$$

$$H^{(l)} = \sigma(Z^{(l)}), \tag{2}$$

where $Z^{(l)}$, $H^{(l)}$, and $W^{(l)}$ represent the intermediate embedding matrix, feature embedding matrix and trainable weight matrix at l-th layer, respectively. σ is a non-linear activation function, like ReLu. P is the propagation matrix that is transformed from the graph adjacency matrix.

During the backward propagation, the gradient of the loss with respect to $W^{(l-1)}$ is as follows:

$$\nabla_{W^{(l-1)}} L = \frac{\partial L}{\partial W^{(l-1)}} = (H^{(l-1)})^T P^T \delta^{(l)}, \tag{3}$$

where $\delta^{(l)}$ denotes the gradient of the loss with respect to $Z^{(l)}$. Then, $W^{(l-1)}$ is updated as follows:

$$W^{(l-1)} = W^{(l-1)} - \eta \nabla_{W^{(l-1)}} L, \tag{4}$$

where η denotes the learning rate of training.

2.2 Spanning Subgraph GNN Training

Given a graph $G = (V, E)$, a spanning subgraph $G_s = (V_s, E_s)$ generated from G is a subgraph with vertex set V, i.e., $V_s = V$ and $E_s \subset E$ [32]. We define edge ratio α between G_s and G is $\frac{|E_s|}{|E|}$. α represents the degree of edge reduction of a

spanning subgraph against the corresponding original graph. The smaller α is, the more edges are deleted, and less memory is demanded for training over the spanning subgraph.

Spanning subgraph GNN training make GNNs only propagation along the subgraph G_s. Therefore, the key GNN operations (Eqs. 1-3) are rewritten as:

$$\tilde{Z}^{(l)} = \tilde{P}H^{(l-1)}W^{(l-1)}, \tag{5}$$

$$\tilde{H}^{(l)} = \sigma(\tilde{Z}^{(l)}), \tag{6}$$

$$\nabla_{W^{(l-1)}}\tilde{L} = \frac{\partial \tilde{L}}{\partial W^{(l-1)}} = (\tilde{H}^{(l-1)})^T \tilde{P}^T \tilde{\delta}^{(l)}, \tag{7}$$

where $\tilde{P}$ is the propagation matrix that is transformed from the spanning subgraph. Spanning subgraph GNN training results in approximated node embedding matrix $\tilde{H}^{(l)}$ and the approximated embedding gradients $\tilde{\delta}^{(l)}$. The model accuracy is affected by these approximated intermediate results as well. We will discuss the main factors that influence the model accuracy in Sect. 4.

3 SpanGNN: Memory-Efficient Full-Graph GNN Learning

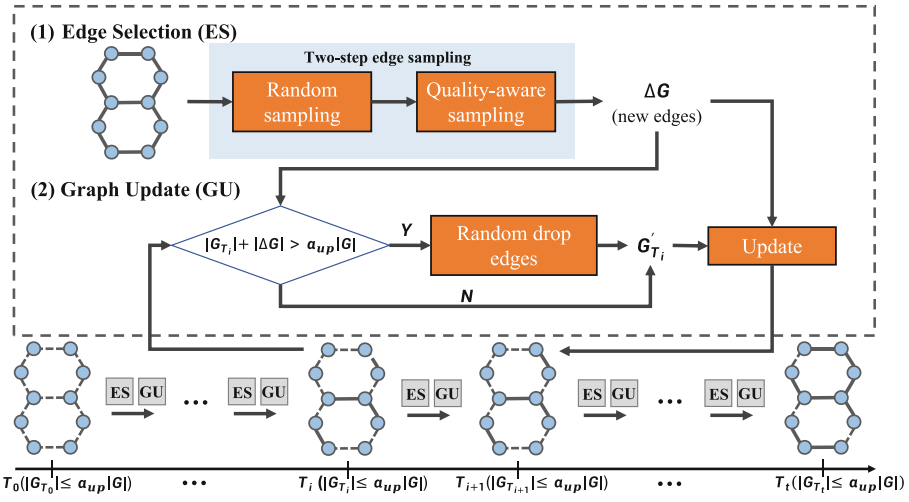


Fig. 1. The framework of SpanGNN.

Figure 1 illustrates the overview of SpanGNN. It starts with an empty spanning subgraph G_{T_0} and progressively includes more edges during the training. For every training epoch T_i, SpanGNN selects a set of edges from the original

graph G, updates the spanning subgraph $G_{T_{i-1}}$ with the selected edge set, and generates a new spanning subgraph G_{T_i}. Furthermore, to limit the peak memory usage, SpanGNN guarantees that, in each training epoch, the edge ratio $\frac{|E_{T_i}|}{|E|}$ does not exceed α_{up}, which is preset by users. According to the definition of edge ratio, the parameter α_{up} implies the upper bound of peak memory usage during the training. Therefore, SpanGNN is able to control the maximal size of peak memory flexibly and is memory-efficient. The pseudocode of the framework is shown in Sect. 2.2 of our technical report [15].

Edge Selection. Edges in the graph contribute differently to the GNN training, so it is important to pick out the most beneficial edges for the training to guarantee the model accuracy. Weighted sampling is a standard approach to select important edges in priority. In this paper, we analyze two types of factors that influence the model accuracy and propose quality-aware edge selection approach in Sect. 4. The new edge selection approach adopts variance-reduced sampling strategy and gradient-noise reduced sampling strategy to select high-quality edges. However, directly sampling from the entire graph with non-uniform probability distribution is time-consuming. We further introduce the two-step edge sampling method to speed up the edge selection. In addition, using the quality-aware edge selection approach, the training process of SpanGNN aligns with the principles of curriculum learning (discussed in Sect. 5), therefore, SpanGNN has high accuracy.

Graph Update. In order to continuously satisfy the edge ratio constraint (i.e., $\frac{|G_{T_i}|}{|G|} \leq \alpha_{up}$), we introduce an edge drop step in the graph update. In each training epoch T_i, if SpanGNN detects that the new spanning subgraph will violate the edge ratio constraint, it first randomly drops a set of edges in $G_{T_{i-1}}$, then adds the selected edge set to the subgraph $G_{T_{i-1}}$; otherwise, the selected edge set is directly added into the subgraph $G_{T_{i-1}}$. The edge drop step helps SpanGNN ensure the memory-efficiency. More importantly, it improves the diversity of the trained spanning subgraphs and enhances the model accuracy as well.

4 Fast Quality-Aware Edge Selection

In this section, we first introduce two types of edge sampling strategies, which are variance-reduced sampling strategy and gradient noise-reduced sampling strategy. Then, we introduce the two-step edge sampling method that optimizes the efficiency of edge selection over large graphs.

4.1 Variance-Minimized Sampling Strategy

The Variance of Aggregated Embedding. Since the edges are probability selected in SpanGNN, the spanning subgraph can be treated as a sampled subgraph from the original graph, and the variance of aggregated embeddings in SpanGNN affects the model accuracy and should not be ignored. Similar to the

existing works [38], the unbiased estimator of aggregated embeddings without activation and the variance of embeddings estimator can be defined as:

$$\xi = \sum_{(l)} \sum_{e} \frac{b_e^{(l)}}{p_e} \mathbb{1}_e^{(l)} \tag{8}$$

$$Var(\xi) = \sum_{e} \frac{(\sum_l b_e^{(l)})^2}{p_e} - \sum_{e} (\sum_{l} b_e^{(l)})^2, \tag{9}$$

where p_e denotes the probability of an edge to be sampled, $b_e^{(l)} = P_{v,u}\tilde{x}_u^{(l-1)} + P_{u,v}\tilde{x}_v^{(l-1)}$, P is the propagation matrix (e.g., the normalized adjacency matrix in GCN), $\tilde{x}$ is feature matrix after linear operation, and $\mathbb{1}_e = 1$ if e is in E_s.

Variance Minimization. To minimize the variance of the aggregated embeddings estimator, we follow the strategy used in GraphSAINT [38]. By using the Cauchy-Schwarz inequality, the variance of aggregated embeddings is minimized when $p_e \propto |\sum_l b_e^{(l)}|$, which can be simplified as:

$$p_e \propto P_{v,u} + P_{u,v} = \frac{1}{deg(u)} + \frac{1}{deg(v)}. \tag{10}$$

The probability p_e defined in Eq. 10 interprets that if two nodes u, v are connected and they have few neighbors, then edge between u and v are more likely to be sampled and to reduce the variance. In other words, such edges will contain more information for the nodes, which is more conducive to the training of the node.

4.2 Gradient Noise-Reduced Sampling Strategy

The Noise of Gradient. As mentioned before, with the spanning subgraphs, the final learned embeddings change compared to the exact ones. Therefore, the results of loss function and gradient change as well. We define the noise of gradient as the change between $\nabla_{W^{(l-1)}} Ls$ that is calculated by original graph training and spanning subgraph training. We formulate the noise of gradient as below:

$$G_{noise} = \nabla_{W^{(l-1)}} \tilde{L} - \nabla_{W^{(l-1)}} L. \tag{11}$$

Gradient Noise Reduction. The noise of gradient slows down the convergence and affects the model accuracy. To solve the problem, we derive a probability distribution for edge sampling that can reduce the upper bound of gradient noise. The probability of an edge $e(u, v)$ is formulated as:

$$p_{v,u} = \frac{\|P_{*,u}\|_2}{\sum_{(v,u)\in E} \|P_{*,u}\|_2}, \tag{12}$$

where $P_{*,u}$ denotes the vector of $node_u$'s propagation matrix.. The larger the sampling probability of the edge, the smaller the gradient noise in the training process.

Next, we theoretically analyze the above probability and dig into the upper bound of the expected gradient noise, which is summarized in Theorem 1.

Theorem 1. ***Upper bound of the expected gradient noise.*** *Given the square of Frobenius norm* $\|P\|_F^2$, $\left\|H^{(l)}\right\|_F^2$, $\left\|\delta^{(l)}\right\|_F^2$ *are bounded by some constants* B, C, D *and the L2-norm* $\left\|H^{(l)}W^{(l)}\right\|$ *is bounded by constant* ξ. *Assume that the activation function* σ *is* $\rho_\sigma-$*Lipschitz and the gradient* $\nabla_{Z^{(l)}}L$ *is* ρ_Z-*Lipschitz, then we have:*

$$E\left[\|G_{noise}\|_F^2\right] \leq (2BD\rho_\sigma + 4BC\rho_Z)E\left[\left\|\tilde{Z}^{(l)} - Z^{(l)}\right\|_F^2\right] + 4BCD. \tag{13}$$

According to E.q. 13, the upper bound of the expected gradient noise is decided by $\left\|\tilde{Z}^{(l)} - Z^{(l)}\right\|_F^2$, i.e., the expected value of the difference of the hidden layer embedding. We further analyze the upper bound of of this difference.

Theorem 2. ***Upper bound of the expected hidden embeddings' difference.*** *Given the entire edge set* E *and the selected edge subset* E_s, *we derive the following inequation:*

$$E_{E_s}\left[\left\|\tilde{Z}_{V,*} - Z_{V,*}\right\|_F^2\right] \leq \frac{1}{|E_s|}\sum_{(v,u)\in E}\frac{1}{p_{(v,u)}}\|P_{*,u}\|_2^2\,\xi^2. \tag{14}$$

The detailed proof of the above two theorems are presented in Sect. 4.2 of our technical report [15]. As illustrated in E.q. 14, we find the upper bound of hidden embeddings' difference is related to edge sampling probability $p_{v,u}$. By combining Eq. 13 and removing the constants that are hard to calculate, we can formulate a gradient noise optimization problem and minimize the value of the noise by using the following constraint:

$$s.t. \sum_{(v,u)\in E} p_{(v,u)} = 1 \tag{15}$$

Based on E.q. 14 and Eq. 15, by using the Lagrange function, we derive the edge sampling probability as defined in E.q. 12, and the probability can reduce the gradient noise in the spanning subgraph training.

4.3 Two-Step Edge Sampling Method

The probabilities defined by Eqs. 10 and 12 are non-uniform. It is a challenge to fast sample non-uniform distribution in large sample space [24]. An efficient sampling method, Alias sampling [28], requires massive memory and entails the high cost of building data structures. In this paper, we propose a simple but effective approximate sampling method – two-step edge sampling to speed up the quality-aware edge selection process.

In the first step, SpanGNN reduces the sample space by randomly sampling an edge set, denoted as e_t, in iteration T_t. This step confines the final selected edges focusing on the edge set e_t rather than the entire edge set E. In the second step, SpanGNN samples $e_t^{'}$ from the edge set e_t according to the probability defined by E.q. 10 and 12. The pseudocode of quality-aware edge selection with two-step sampling is given in Sect. 4.3 of our technical report [15]. Additionally, we show more details about parameter sensitive analysis of two-step sampling in Sect. 6.6 of our technical report [15].

Here, we discuss the advantages of two-step sampling with a memory-efficient non-uniform sampling method, which first constructs a cumulative probability array, then uses random numbers to select elements. The time complexity entailed by the first step of random sampling from the entire edge set is $O(|e|)$. The time complexity of the second step of weighted sampling is about $O(|e| + |e^{'}|log(|e|))$. Therefore, the total time complexity of the two-step sampling is $O(|e|) + O(|e^{'}|log(|e|))$. However, if we directly sample $|e^{'}|$ edges from entire edge set, the time complexity is $O(|E| + |e^{'}|log(|E|))$. In practice, $|e|$ is typically several to ten times $|e^{'}|$, while $|E|$ can be up to a hundred times larger than $|e|$. Therefore, the time cost of the two-step sampling is lower than that of direct sampling.

5 Connection to Curriculum Learning

In this section, we analyze the connection between SpanGNN and curriculum learning. To our knowledge, curriculum learning increases the robustness of the learned model against noisy training samples by training samples from easy to hard. An intuitive explanation is that curriculum learning spends less time with the harder (noisy) samples to achieve better robustness.

SpanGNN incorporates the principles of curriculum learning by constructing different graph structures (i.e., spanning subgraphs) during the learning process. SpanGNN not only mirrors the educational strategy of progressing from easy to hard lessons, but also aligns with the model's need to first grasp fundamental concepts before tackling more challenging tasks. The detailed discussion is as follows.

First, in SpanGNN, the empty graph at the beginning can be regarded as the simplest 'course'. In the training process, edges are gradually added to the graph. This progressive learning process helps the model master basic structural information first, and then learn more complex graph structures, which helps the model to learn more robust and avoid overfitting.

Second, through the Quality-aware Edge Selection, we prioritize edges that are more significant for model training, to help minimize feature variance and reduce gradient noise. Edges with smaller feature variance mean that the aggregated features are more consistent. Also, edges with less gradient noise mean that they can help the model learn more stable. This is similar to the 'from easy to hard' in curriculum learning, where we initially learn the data that will be more beneficial for subsequent learning.

6 Experimental Studies

In this section, we start with the descriptions of experimental settings, which cover the datasets and configurations used in the experiments. Then, we evaluate the performance of SpanGNN by comparing it with full-graph training methods, conduct ablation studies to verify the effectiveness of the proposed techniques, and study the efficiency of SpanGNN. Finally, we also compare SpanGNN with mini-batch training methods to demonstrate that generally SpanGNN is able to achieve high accuracy. In addition, due to the limited space, we put the results of parameter sensitivity in Sect. 6.6 of our technical report [15].

6.1 Experimental Setups

Table 1. Dataset statistics

Dataset	Dataset attributions			
name	*#Nodes*	*#Edges*	*Features*	*Classes*
Ogbn-proteins	132,534	79,122,504	8	112
Reddit	232,965	114,615,892	602	41
Amazon	1,598,960	264,339,468	200	107
Ogbn-products	2,449,029	126,167,053	100	47

Environments and Datasets. We implemented SpanGNN with PyTorch 2.0.1, and the code is released[1]. We evaluate the performance of SpanGNN using two common GNN models including GCN [19] and SAGE [16] with the mean aggregator. All experiments are conducted on NVIDIA RTX A6000. We use four large graph datasets. Table 1 lists the summary of the datasets.

Performance Metrics and Evaluation Protocol. Accuracy is used to measure the effectiveness of SpanGNN on Reddit and Ogbn-products datasets, F1-score is used on Amazon, and AUC-ROC is used on Ogbn-proteins. All performance metrics are calculated on the validation set and the results are the average of three times experiments. Furthermore, we conduct experiments under different edge ratios to verify the memory-efficiency of SpanGNN.

Baselines. 1) **Full-graph**. It is a naive full-graph training method, but consumes heavy GPU memory. 2) **DropEdge**. It has good scalability for training on large graphs. 3) **GraphSAGE**, **ClusterGCN** and **GraphSAINT** are selected as the representations of the mini-batch training.

Additionally, SpanGNN equipped with variance-minimized sampling and gradient noise-reduced sampling respectively are denoted by **SpanGNN-F** and **SpanGNN-G**.

[1] https://github.com/guxizhi/SpanGNN.

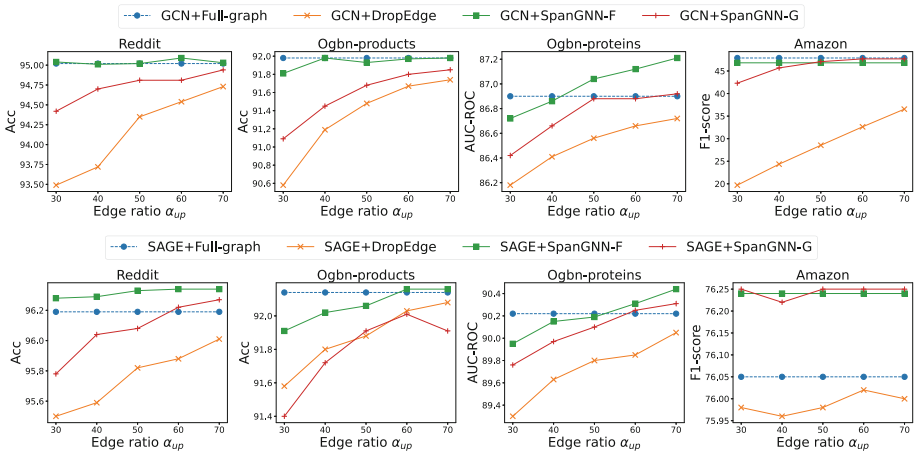


Fig. 2. The performance of the training methods on GCN (Up-side) and SAGE (Down-side) with various edge ratios.

6.2 Performance of SpanGNN

Comparison of Model Accuracy. Figure. 2 illustrates the model accuracy of SpanGNN, Full-graph and DropEdge on GCN and SAGE with various edge ratios α_{up}, which are set from 0.3 to 0.7. We see that SpanGNN-F's performance is similar to or even better than Full-graph. For example, on Reddit, the accuracy of SpanGNN-F is higher than the one of Full-graph with SAGE, regardless of α_{up}. On Ogbn-proteins, as α_{up} gets larger, the AUC-ROC of SpanGNN-F gradually exceeds the one of Full-graph. The exception is on Amazon, where SpanGNN-G is better than SpanGNN-F as α_{up} gets larger. This is because SpanGNN-F's sampling probability on Amazon is extremely skewed, and it is caused by the fact that few edges are connected by two low-degree nodes. These minority edges are given larger weight during selection, causing them to be selected repeatedly in every edge selection. It is hard to obtain enough edges for the spanning subgraph (i.e., the edge ratio of a spanning subgraph is hard to reach α_{up}) and results in a decrease of F1-score. This problem also reduces the size of peak memory usage. In Fig. 3, we see that the peak memory usage of SpanGNN-F is stable with respect to different edge ratios on Amazon.

Compared to SpanGNN, DropEdge suffers from the decrease in model performance more seriously. On Reddit, DropEdge losses the accuracy by up to 1.5% on GCN and by up to 0.8% on SAGE. Even worse, DropEdge severely damages the model's F1-score on Amazon by more than 25%. Overall, SpanGNN is better at ensuring model's performance compared to DropEdge.

Comparison of Peak Memory Usage. Here we compare the peak memory usage among SpanGNN, Full-graph, DropeEdge. As shown in Fig. 3, we see that reducing the number of edges effectively reduces the size of peak memory by

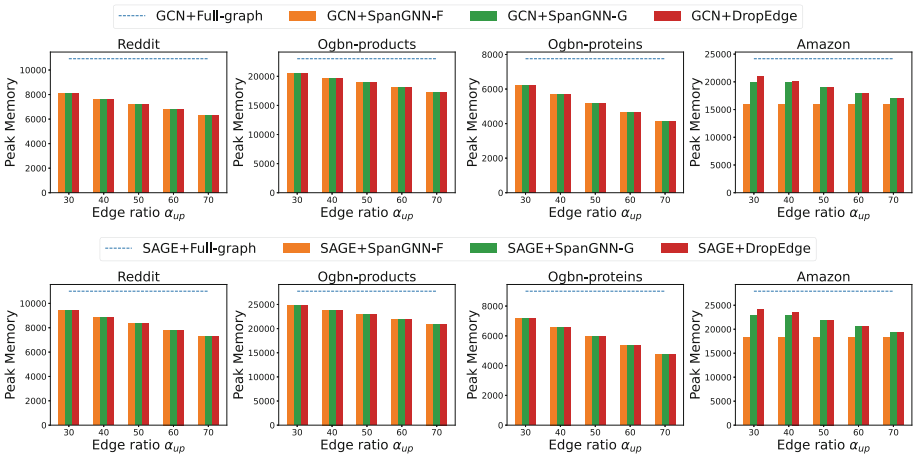


Fig. 3. Peak Memory Usage on GCN(Up-side) and SAGE(Down-side).

comparing SpanGNN and Full-graph. There is no significant difference between SpanGNN and DropEdge, since they drop the same size of edges. In addition, the percentage of peak memory saved is also independent of the model. By using only 30% edges, SpanGNN and DropEdge can reduce the peak memory usage by 42%, 25% and 47% on Reddit, Ogbn-products and Ogbn-proteins, respectively. Note that on Amazon, due to the actual edge ratio cannot achieve α_{up}, which is discussed in the "Comparison of model accuracy", SpanGNN-F has less peak memory overhead.

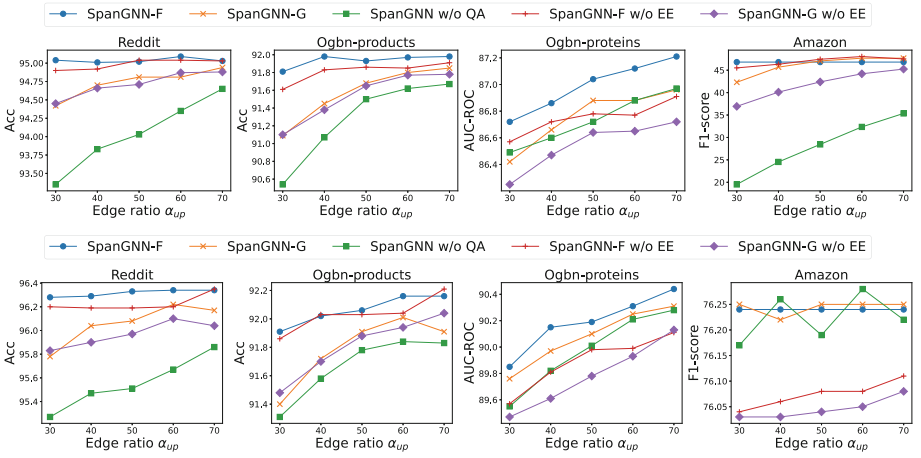


Fig. 4. Ablation studies on GCN(Up-side) and SAGE(Down-side)

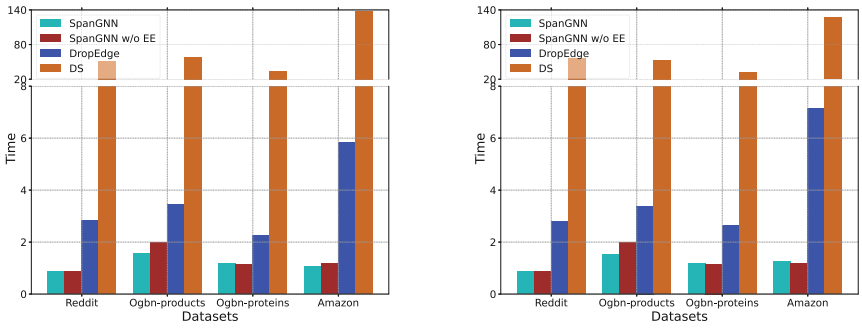
6.3 Ablation Studies

Effectiveness of the Framework. In order to verify the effectiveness of the principles of curriculum learning used by SpanGNN, we compare the model accuracy between SpanGNN and SpanGNN w/o EE. Instead of the empty graph, SpanGNN w/o EE is initialized by the graph with $\alpha_{up}|G|$ edges, which are selected by quality-aware edge selection. The results are shown in Fig. 4.

The results indicate that SpanGNN improves the model performance on various datasets by adopting the curriculum learning principles. On Ogbn-proteins, it is clear that SpanGNN outperforms SpanGNN w/o EE and the improvement is around 0.2%. On other datasets, depending on the based model and the value of α_{up}, SpanGNN is generally better than or equal to SpanGNN w/o EE.

Effectiveness of Quality-Aware Edge Selection. In order to verify the effectiveness of variance-minimized sampling and gradient noise-reduced sampling strategies, we compare the model performance among SpanGNN-G, SpanGNN-F, and SpanGNN w/o QA. Here SpanGNN w/o QA applies random sampling instead of quality-aware sampling. The results are shown in Fig. 4.

Generally, SpanGNN-G and SpanGNN-F have better performance than SpanGNN w/o QA. The advantage can reach 1.5% on Reddit and even more than 20% on Amazon. In certain cases, SpanGNN w/o QA might outperform SpanGNN. As discussed in the "Comparison of model accuracy", on Amazon, the spanning subgraph in SpanGNN is easy to contain fewer edges than the required ones defined by α_{up} because of the skewed sampling probability. However, SpanGNN w/o QA can successfully reach α_{up}, and contain sufficient edges. Therefore, SpanGNN does not always perform better than SpanGNN w/o QA on SAGE. Overall, we conclude that the variance-minimized sampling and the gradient noise-reduced sampling generally plays important roles in improving performance of SpanGNN.



(a) GCN (b) SAGE

Fig. 5. The average time cost of generating spanning subgraphs.

6.4 Efficiency of SpanGNN

In this section, we demonstrate the efficiency of SpanGNN by comparing the average time cost of generating spanning subgraphs. Figure 5 illustrates the results of SpanGNN, SpanGNN w/o EE, DropEdge and direct sampling from the entire graph based on quality-aware edge selection (DS). To guarantee the fairness of the comparison, all methods have the same edge ratio (i.e., $\alpha_{up} = 0.3$).

As we can see, SpanGNN is more efficient than other frameworks, and integrating curriculum learning principle does not destroy the execution efficiency. Specifically, compared to DropEdge, SpanGNN speeds up from 1.95x to 5.65x on different datasets. As analyzed in Sect. 4.3, the time complexity of generating spanning subgraphs in SpanGNN is proportional to the number of first-step sampling edges e. Due to dropping a lot of edges each iteration (i.e., $\alpha_{up} = 0.3$), DropEdge entails much more time cost than SpanGNN. When compared to DS, SpanGNN speeds up from 26.99x to 132.08x. This is because the time complexity of DS is proportional to the number of edges in the entire graphs, which can reach hundreds of times that of e.

Table 2. The comparison of model performance with mini-batch training methods. Note that SpanGNN's results are determined by taking the best one among different edge ratios.

GNN	Model	Reddit	Ogbn-products	Amazon	Ogbn-proteins
		Acc	Acc	F1-score	AUC-ROC
GCN	SpanGNN-G	95.26	91.50	**47.79**	87.11
	SpanGNN-F	95.46	**91.68**	46.78	**87.19**
	GraphSAGE	91.99	90.18	28.73	71.31
	ClusterGCN	92.05	89.94	46.86	79.30
	GraphSAINT	**96.53**	90.14	7.50	80.12
SAGE	SpanGNN-G	96.51	91.33	76.29	90.36
	SpanGNN-F	96.62	**91.90**	76.26	**90.49**
	GraphSAGE	94.55	90.57	72.99	82.35
	ClusterGCN	94.74	90.55	**77.43**	83.54
	GraphSAINT	**97.46**	90.15	75.21	85.35

6.5 Performance of SpanGNN Compared to Mini-batch Training

In this section, we compare SpanGNN with different mini-batch training methods in terms of model performance. The memory usages of mini-batch training is related to the batch size, and it is more flexible than SpanGNN in terms of memory consumption. However, as shown in Table 2, generally SpanGNN achieves better performance than the mini-batch methods. On Ogbn-products

and Ogbn-proteins, SpanGNN always outperforms the mini-batch methods and its improvements can achieve 1.7% and 7.0% respectively. On other datasets, SpanGNN is either the best one or the second best but very close to the best one. Therefore, compared to the mini-batch training, SpanGNN achieves high model performance.

7 Related Work

7.1 Memory-Efficient Graph Neural Networks

Mini-batch training is an effective approach to reduce the memory consumption. Existing works do a lot of exploration on sampling methods with mini-batch training approach. The works [5,16] apply node-level sampling to select a set of nodes in neighbors. In this way, it reduce the number of each node's neighbors in the phase of aggregation. However, it can not resolve the problem of 'neighbor explosion' when GNNs goes deeper. The works [8,42] apply layer-level sampling to select a fixed number of nodes in each GNN layer. Since this type of sampling methods use fixed number of nodes in the each layer, it can alleviate the 'neighbor explosion'. However, FastGCN [8] suffers from unbalanced receptive fields. LADIES [42] tracks each node's neighbors in the previous layer and calculates an importance estimator, but causes much overhead. The works [6,38] apply subgraph-level sampling to limit the aggregation field to a subgraph. ClusterGCN [6] partitions the graph into a set of clusters and then randomly combines partitions to be a mini-batch. GraphSAINT [38] directly forms subgraphs with overlapping nodes among mini-batches. However, compared to Full-graph training, mini-batch training incurs information loss.

Even though almost no work explicitly discusses using spanning subgraph to reduce peak memory usage during GNNs training, there exist some close works. DropEdge [22] randomly removes a certain percentage of edges from the original input graph in each epoch. It alleviates the problem of over-smoothness [2,14] and over-fitting and can be considered as a strategy that uses spanning subgraph. TADropEdge [12] additionally considers the factors of graph structure. They analyze the graph connectivity and gives larger weight to keep inter-cluster edges in GNNs training. NeuralSparse [41] applies a deep neural network to learn how to sparse graphs with the feedback of downstream prediction tasks. It improves generalization ability by removing potentially task-irrelevant edges. SGCN [20] also considers sparsification as an optimization problem and applies ADMM-based solution to solve it. But these works focus on improving the prediction results, overlooking the problem of peak memory usage. In addition, some other works directly delete or change the structure of GNNs model. SGC [33] reduces redundant calculations by deleting the non-linear activation function between GCN layers. PPRGo [4], by calculating the influence matrix, avoids the overhead of collecting multi-hop neighbors. However, they fundamentally change the characteristics of GNNs, and cannot directly be applied to existing models.

7.2 Curriculum Learning on GNN

Recently, curriculum learning is introduced into GNNs training and achieves performance improvement in GNN models. CurGraph [29] introduces curriculum learning to train GNNs with graphs in ascending order of difficulty. This method uses the informax technique for graph-level embeddings and a neural density estimator to model the embedding distributions. After calculating the difficulty scores of graphs, it first exposes GNN models to easy graphs and moves on to harder ones. They focus on the prediction of graphs. CLnode [31] defines the difficulty of samples at the level of the node and applies various pacing functions to train GNNs from easy-to-hard. It measures nodes' difficulty from the perspective of neighborhoods and features. RCL [40] considers that connections of nodes significantly affect the curriculum learning. It distinguishes the level of difficulty for edges and gradually incorporates more information at the level of edges. However, neither CLnode nor RCL takes the memory usage into account, and they need to train GNNs on the entire graphs. Differently, our work sets an upper bound of the number of edges and designs difficulty scoring function by fully considering the impact of the spanning subgraph.

8 Conclusion

In this paper, we proposed SpanGNN that carries out GNNs training on large-scale graphs efficiently by using spanning subgraphs and integrating the principles of curriculum learning. SpanGNN consists of two main components to limit the memory overhead and ensure the model performance. Quality-aware edge selection samples beneficial edges for spanning subgraph GNN training and follows the manner of curriculum learning to add edges for training. Graph update determines the size of the spanning subgraph at each epoch to control the peak memory. Overall, we provide an efficient large-scale GNN training method that can reduce memory overhead and maintain the model performance.

Acknowledgement. This work is supported by the National Natural Science Foundation of China (Nos. 62272054, 62192784), Beijing Nova Program (No. 20230484319), and Xiaomi Young Talents Program. Lei Chen's work is partially supported by National Science Foundation of China (NSFC) under Grant No. U22B2060, the Hong Kong RGC GRF Project 16213620, RIF Project R6020-19, AOE Project AoE/E-603/18, Theme-based project TRS T41-603/20R, CRF Project C2004-21G, China NSFC No. 61729201, Guangdong Basic and Applied Basic Research Foundation 2019B151530001, Hong Kong ITC ITF grants MHX/078/21 and PRP/004/22FX, Microsoft Research Asia Collaborative Research Grant and HKUST-Webank joint research lab grants.

References

1. Ahmedt-Aristizabal, D., Armin, M.A., Denman, S., Fookes, C., Petersson, L.: Graph-based deep learning for medical diagnosis and analysis: Past, present and future. Sensors **21**(14), 4758 (2021)
2. Bause, F., Moustafa, S., Langguth, J., Gansterer, W.N., Kriege, N.M.: On the Two Sides Of Redundancy in Graph Neural Networks (2024)
3. Bengio, Y., Louradour, J., Collobert, R., Weston, J.: Curriculum learning. In: ICML, pp. 41–48 (2009)
4. Bojchevski, A., et al.: Scaling graph neural networks with approximate pagerank. In: KDD, pp. 2464–2473 (2020)
5. Chen, J., Zhu, J., Song, L.: Stochastic training of graph convolutional networks with variance reduction. In: ICML, vol. 80, pp. 942–950 (2018)
6. Chiang, W.L., Liu, X., Si, S., Li, Y., Bengio, S., Hsieh, C.J.: Cluster-GCN: an efficient algorithm for training deep and large graph convolutional networks. In: KDD, pp. 257–266 (2019)
7. Choi, E., et al.: Learning the graphical structure of electronic health records with graph convolutional transformer. In: AAAI, pp. 606–613 (2020)
8. Defferrard, M., Bresson, X., Vandergheynst, P.: Convolutional neural networks on graphs with fast localized spectral filtering. In: NIPS, pp. 3844–3852 (2016)
9. Duvenaud, D.K., et al.: Convolutional networks on graphs for learning molecular fingerprints. NIPS **28** (2015)
10. Fan, W., et al.: Graph neural networks for social recommendation. In: WWW, pp. 417–426 (2019)
11. Fout, A., Byrd, J., Shariat, B., Ben-Hur, A.: Protein interface prediction using graph convolutional networks. In: NIPS. **30**, 6533–6542 (2017)
12. Gao, Z., Bhattacharya, S., Zhang, L., Blum, R.S., Ribeiro, A., Sadler, B.M.: Training robust graph neural networks with topology adaptive edge dropping. arXiv preprint arXiv:2106.02892 (2021)
13. Gilmer, J., Schoenholz, S.S., Riley, P.F., Vinyals, O., Dahl, G.E.: Neural message passing for quantum chemistry. In: ICML, pp. 1263–1272 (2017)
14. Graziani, C., Drucks, T., Bianchini, M., franco scarselli, Gärtner, T.: No PAIN no gain: more expressive GNNs with paths. In: NeurIPS 2023 Workshop: New Frontiers in Graph Learning (2023). https://openreview.net/forum?id=q2xXh4M9Dx
15. Gu, X., Li, H., Gao, S., Zhang, X., Chen, L., Shao, Y.: SpanGNN: towards memory-efficient graph neural networks via spanning subgraph training (2024)
16. Hamilton, W., Ying, Z., Leskovec, J.: Inductive representation learning on large graphs. NIPS **30** (2017)
17. Huang, T., Dong, Y., Ding, M., Yang, Z., Feng, W., Wang, X., Tang, J.: MixGCF: an improved training method for graph neural network-based recommender systems. In: KDD, pp. 665–674 (2021)
18. Jia, Z., Lin, S., Gao, M., Zaharia, M., Aiken, A.: Improving the accuracy, scalability, and performance of graph neural networks with roc. In: MLSys **2**, 187–198 (2020)
19. Kipf, T.N., Welling, M.: Semi-supervised classification with graph convolutional networks. arXiv preprint arXiv:1609.02907 (2016)
20. Li, J., Zhang, T., Tian, H., Jin, S., Fardad, M., Zafarani, R.: SGCN: a graph sparsifier based on graph convolutional networks. In: PAKDD, pp. 275–287 (2020)
21. Rao, J., Zhou, X., Lu, Y., Zhao, H., Yang, Y.: Imputing single-cell RNA-seq data by combining graph convolution and autoencoder neural networks. Iscience **24**(5), 102393 (2021)

22. Rong, Y., Huang, W., Xu, T., Huang, J.: DropEdge: towards deep graph convolutional networks on node classification. In: ICLR (2019)
23. Schlichtkrull, M., Kipf, T.N., Bloem, P., Van Den Berg, R., Titov, I., Welling, M.: Modeling relational data with graph convolutional networks. In: ESWC, pp. 593–607 (2018)
24. Shao, Y., Huang, S., Li, Y., Miao, X., Cui, B., Chen, L.: Memory-aware framework for fast and scalable second-order random walk over billion-edge natural graphs. VLDB J. **30**(5), 769–797 (2021)
25. Sun, C., Ning, Y., Shen, D., Nie, T.: Graph neural network-based short-term load forecasting with temporal convolution. Data Sci, Eng. **9**(2), 113–132 (2023)
26. Tan, Q., Liu, N., Hu, X.: Deep representation learning for social network analysis. Frontiers Big Data **2**, 2 (2019)
27. Veličković, P., Cucurull, G., Casanova, A., Romero, A., Liò, P., Bengio, Y.: Graph attention networks. In: ICLR, pp. 1–12 (2018)
28. Walker, A.J.: An efficient method for generating discrete random variables with general distributions. TOMS **3**(3), 253–256 (1977)
29. Wang, Y., Wang, W., Liang, Y., Cai, Y., Hooi, B.: CurGraph: curriculum learning for graph classification. In: WWW, pp. 1238–1248 (2021)
30. Wang, Z., Wang, Y., Yuan, C., Gu, R., Huang, Y.: Empirical analysis of performance bottlenecks in graph neural network training and inference with GPUs. Neurocomputing **446**, 165–191 (2021)
31. Wei, X., Gong, X., Zhan, Y., Du, B., Luo, Y., Hu, W.: CLNode: curriculum learning for node classification. In: WSDM, pp. 670–678 (2023)
32. West, D.B.: Introduction to Graph Theory. Prentice Hall, 2 edn. (September 2000)
33. Wu, F., Souza, A., Zhang, T., Fifty, C., Yu, T., Weinberger, K.: Simplifying graph convolutional networks. In: ICML, pp. 6861–6871 (2019)
34. Wu, S., Sun, F., Zhang, W., Xie, X., Cui, B.: Graph neural networks in recommender systems: a survey. Comput. Surv. **55**(5), 1–37 (2022)
35. Xiao, S., Zhu, D., Tang, C., et al.: Combining graph contrastive embedding and multi-head cross-attention transfer for cross-domain recommendation. Data Sci. Eng. **8**, 247–262 (2023). https://doi.org/10.1007/s41019-023-00226-7
36. Xu, K., Hu, W., Leskovec, J., Jegelka, S.: How powerful are graph neural networks? In: ICLR, pp. 1–17 (2019)
37. Ying, Z., You, J., Morris, C., Ren, X., Hamilton, W., Leskovec, J.: Hierarchical graph representation learning with differentiable pooling. NIPS **31** (2018)
38. Zeng, H., Zhou, H., Srivastava, A., Kannan, R., Prasanna, V.K.: Graphsaint: Graph sampling based inductive learning method. In: ICLR. OpenReview.net (2020)
39. Zhang, M., Chen, Y.: Link prediction based on graph neural networks. NIPS **31**, 1–11 (2018)
40. Zhang, Z., Wang, J., Zhao, L.: Curriculum learning for graph neural networks: which edges should we learn first. NIPS **36** (2024)
41. Zheng, C., et al.: Robust graph representation learning via neural sparsification. In: ICML, pp. 11458–11468. PMLR (2020)
42. Zou, D., Hu, Z., Wang, Y., Jiang, S., Sun, Y., Gu, Q.: Layer-dependent importance sampling for training deep and large graph convolutional networks. In: NIPS, pp. 1–11 (2019)