Abstract—Graph neural networks (GNNs) tend to suffer from high computation costs due to the exponentially increasing scale of graph data and a large number of model parameters, which restricts their utility in practical applications. To this end, some recent works focus on sparsifying GNNs (including graph structures and model parameters) with the lottery ticket hypothesis (LTH) to reduce inference costs while maintaining performance levels. However, the LTH-based methods suffer from two major drawbacks: 1) they require exhaustive and iterative training of dense models, resulting in an extremely large training computation cost, and 2) they only trim graph structures and model parameters but ignore the node feature dimension, where vast redundancy exists. To overcome the above limitations, we propose a comprehensive graph gradual pruning framework termed CGP. This is achieved by designing a during-training graph pruning paradigm to dynamically prune GNNs within one training process. Unlike LTH-based methods, the proposed CGP approach requires no retraining, which significantly reduces the computation costs. Furthermore, we design a cosparsifying strategy to comprehensively trim all the three core elements of GNNs: graph structures, node features, and model parameters. Next, to refine the pruning operation, we introduce a regrowth strategy to reestablish the pruned but important connections. The proposed CGP is evaluated over a node classification task across six GNN architectures, including shallow models (graph convolutional network (GCN) and graph attention network (GAT)), shallow-but-deep-propagation models [simple graph convolution (SGC) and approximate personalized propagation of neural predictions (APPNP)], and deep models [GCN via initial residual and identity mapping (GCNI) and residual GCN (ResGCN)], on a total of 14 real-world graph datasets, including large-scale graph datasets from the challenging Open Graph Benchmark (OGB). Experiments reveal that the proposed strategy greatly improves both training and inference efficiency while matching or even exceeding the accuracy of the existing methods.

Index Terms—Dynamic sparse training, gradual pruning, graph neural networks (GNNs), homophily and heterophily, node classification.

I. INTRODUCTION

Graph neural networks (GNNs) [1], [2], [3], [4] have achieved notable success in a variety of applications [5], [6], [7], [8] and have consequently become a rapidly growing area of research [9], [10], [11], [12]. Despite success, GNNs exhibit exponentially growing computation costs with the size of the graph data [13] and the complexity of the model structure increasing [14], [15], [16], during both training and inference [17]. This has proved prohibitive to their deployment in resource-constrained or time-sensitive applications. For example, a two-layer graph convolutional network (GCN) model [1] with 32 hidden units requires approximately 19 GFLOPs (flops: floating point operations) to process the Reddit graph [18], [19], twice as much as the requirements of the ResNet50 model [20] on ImageNet. If the number of layers of the GCN model increases to 64, the model requires approximately 73 GFLOPs. Such enormous computation costs of GNNs are primarily due to three aspects: 1) the large-scale adjacency matrix in real-world datasets (e.g., millions of edges in the OGB datasets [13]); 2) the high-dimensional node feature vectors (e.g., each node in the Citeseer graph has 3703 features); and 3) the sheer number of model parameters (e.g., a 64-layer GCN via initial residual and identity mapping (GCNI) [16] with 64 hidden units contains about 262,144 parameters).

To reduce these enormous computation costs, several approaches (i.e., unified GNN sparsification (UGS) [21], graph early-bird tickets (GEBT) [19], and inductive co-pruning of GNNs (ICPG) [22]) have been developed based on the lottery ticket hypothesis (LTH) [23]. Specifically, these methods propose to generalize the popular LTH to GNNs to find a subnetwork and a subgraph that best preserve the GNN

---

Comprehensive Graph Gradual Pruning for Sparse Training in Graph Neural Networks

Chuang Liu, Xueqi Ma, Yibing Zhan, Member, IEEE, Liang Ding, Member, IEEE, Dapeng Tao, Member, IEEE, Bo Du, Senior Member, IEEE, Wenbin Hu, Member, IEEE, and Danilo P. Mandic, Fellow, IEEE
performance in a sparse manner. The LTH-based approaches usually adopt an iterative cycle (train-then-sparsify), that is, 1) train-dense; 2) prune; and 3) retrain, focusing on sparsifying edges and model weights for the principal purpose of reducing the inference cost.

Despite their good performance, the above LTH-based sparsification methods are still unsatisfactory due to the following reasons.

1) Enormous Training Costs: The existing methods necessitate training dense models exhaustively and iteratively (up to $20\times$),\(^1\) which considerably increases the training time [see Fig. 1 (left)]. In addition, these methods cannot be implemented on low-capacity devices since the dense models must be stored in memory at the outset of the operation.

2) Redundancy of Node Features: The existing techniques only coparsify the input graph structure (i.e., adjacency matrix) and model parameters, thereby neglecting the benefits of pruning the high-dimensional node feature, which is a fundamental but redundant element in most GNNs.

To solve the aforementioned issues, we introduce a comprehensive graph gradual pruning framework, termed CGP, a novel and efficient sparse training framework for GNNs, whereby 1) CGP uses a during-training graph pruning paradigm that enables the discovery of high-performing sparse GNNs in a single training procedure [see Fig. 1 (right)], as opposed to the LTH iterative pruning and retraining; 2) we leverage the unique structures of GNNs to comprehensively exploit GNN-specific pruning strategies, containing graph structures, node features, and model parameters, to sufficiently boost the training and inference efficiency of GNNs; and 3) we incorporate regrowth schemes into our framework to recover the “mistakenly” pruned connections (i.e., the significant connections that are pruned during the pruning process). In addition, CGP supports sparse-to-sparse training, which enables sparse initializations and maintains sparse values throughout training. This further accelerates the overall training process, hence facilitating the development of GNNs in real-time or source-constrained applications.

To evaluate the effectiveness of CGP, extensive experiments are conducted across six GNN models on 14 benchmark graph datasets. We demonstrate that our method reliably replicates the performance levels of the LTH-based methods while achieving significantly faster inference (e.g., up to $250\times$ faster inference on the Photo dataset). In particular, on high-heterophily benchmark datasets, our method consistently outperforms state-of-the-art (SOTA) hetero-GNN methods and achieves an improvement of up to 120% over baseline GNN models (GCN [1], GraphSAGE [2], graph attention network (GAT) [3], and graph isomorphism network (GIN) [4]), is to update the embedding of each node with messages from its neighboring nodes. Despite enormous success, the above GNN models are shallow and achieve their best performance with two-layer models. Such shallow architectures limit the expressive ability, hindering the great potential of GNNs in large-scale real-world graphs [13]. Therefore, there is an increasing trend to design deeper GNN architectures such as residual GCN (ResGCN) and GCNII. The ResGCN [14], [15] borrows concepts from CNNs, including residual connections and dilated convolutions and adapts them to build an extremely deep (56-layer) GCN model. The GCNII [16] proposes to deepen GNN models to 64 layers with two simple techniques: initial residual and identity mapping. For an in-depth understanding of deeper GNNs, refer to [27]. Despite the impressive empirical results that deeper GNNs have shown on some large-scale graphs, they still suffer from high computation costs due to the increasing scale of graph data and model parameters used in practice. Consequently, the high computation costs limit their deployment in resource-constrained or real-time applications.

B. Graph Sparsification and Coarsening

Graph sparsification and coarsening are two ways to reduce the size of a graph to speed up graph training. Graph sparsification typically removes the task-irrelevant edges in a graph, while graph coarsening reduces the number of nodes.

1) Graph Sparsification: Conventional graph sparsification methods [28], [29] are usually unsupervised, and thus, the resulting sparsified graphs aim to preserve graph properties and may not favor downstream tasks. Subsequently, NeuralSparse [30] proposes a supervised graph sparsification technique, which improves generalization power by learning to remove potentially task-irrelevant edges from input graphs. Similar to NeuralSparse, DropEdge [31] is designed to randomly remove edges in each training epoch, which helps mitigate the over-smoothing issue in the training of deep GNNs. Furthermore, SGCN [32] attempts to sparsify the adjacency matrix with the help of alternating direction method of multipliers (ADMM) optimization [33].
Fig. 1. Proposed CGP method versus LTH-based sparsification methods. Left: LTH-based iterative pruning methods. Right: Our proposed gradual pruning method, where $T$ refers to a whole training process when the model is trained to converge. The detailed definitions of $W$, $A$, and $X$, whose pruned values are $W'$, $A'$, and $X'$, respectively, can be found in Section III-A. Note that “iterative” and “gradual” usually have the same meaning [24]. In this article, the iterative pruning refers to the multistep pruning occurring across multitask processes (left), while the gradual pruning refers to the multistep pruning occurring within a single training process (right).

2) Graph Coarsening: Current works on graph coarsening mainly focus on preserving different properties, which are related to the spectrum of the original graph and coarse graph [34], [35]. In addition, graph coarsening refinement network (GOREN) [36] was the first to incorporate GNNs into graph coarsening. Furthermore, Huang et al. [37] attempt to leverage graph coarsening to speed up the training of GNNs in the semisupervised setting. For a comprehensive description of graph sparsification and coarsening, refer to [17] and [38]. However, all the above methods mainly focus on the sparsification of graph structures but ignore the model parameters.

C. GNNs’ Simplification

Recent literature has argued that the training efficiency of GNNs can be further promoted by simplifying redundant operations, despite their current outstanding performance. For example, SGC [25] and scalable inception graph network (SIGN) [39] remove the nonlinear activation [i.e., rectified linear unit (ReLU)] between each layer and precompute the feature aggregation to accelerate training and inference of GNNs. Obviously, these methods lose the powerful expression ability of nonlinear architectures. The PPNP (APPNP) [26] and deep adaptive GNN (DAGNN) [40] decouple the transformation and propagation. In this way, the propagation process can be precomputed, and thus, the classifier (transformation) can be trained faster. PPRGo [41] performs less aggregation with Top-$k$ selection to accelerate the inference of GNNs. Different from the above methods which focus on modifying the model architectures, our CGP sparsifies input graphs and GNN model weights to accelerate GNNs. Moreover, our method is orthogonal to the above models, and we conducted experiments with SGC [42] and APPNP [26], as presented in Section IV.

D. LTH-Based GNN Sparsification

The LTH [23] states that a sparse subnetwork exists in a dense randomly initialized network, called the winning ticket, which can be trained to achieve comparable performance to the dense network with the same weight initialization. The LTH has been developed and extended to different fields [43], [44], [45], [46], [47], [48], [49], [50], [51]. Recently, Chen et al. [21] extend LTH to GNNs and propose the graph lottery ticket (GLT), which refers to cosparsifying input graphs and model parameters. Then, You et al. [19] demonstrate that the Early-Bird Tickets Hypothesis [45] also holds for GNNs and further develop efficient and effective detectors to automatically identify Early-Bird Tickets of GNNs. However, the above two GNN sparsification methods cannot be applied in the inductive learning setting due to the transductive nature of masks. Therefore, Sui et al. [22] propose the ICPG framework to endow GLT with the inductive pruning capacity by adopting an AutoMasker strategy. However, the LTH-based methods necessitate fully and iteratively (up to 20$	imes$) training the dense models, which causes increased computation costs for additional rounds of GNN training. Therefore, to boost both training and inference efficiency, we propose a novel pruning framework to dynamically prune the network parameters in the training stage, which greatly reduces the computational cost and achieves comparable or even better performance compared with the LTH-based methods.

III. METHODOLOGY

A. Preliminaries

1) Notations: A graph $G$ can be represented by an adjacency matrix $A \in \{0, 1\}^{n \times n}$ and a node feature matrix $X \in \mathbb{R}^{n \times d}$, where $n$ is the number of nodes, $d$ is the dimension of node features, and $A[i, j] = 1$ if there exists an edge between node $v_i$ and node $v_j$; otherwise, $A[i, j] = 0$.

2) Graph Neural Networks: The basic idea behind GNN models is to update the embedding of each node with messages from its neighboring nodes. Taking GCN [1] as an example, a two-layer GCN model can be formulated as

$$ Z = f(A, X) = \text{softmax}(\hat{A} \text{ReLU}(\hat{A}XW^{(0)})W^{(1)}) $$

where $\hat{A} = \hat{D}^{-1/2}(A + I_n)\hat{D}^{-1/2}$ is the adjacency matrix normalized by the degree matrix $\hat{D}$ of $A + I_n$, $Z$ is the matrix of GCN’s predictions, $W = (W^{(0)}, W^{(1)})$ is the weights of the two-layer GCN model, and ReLU is an activation function.

B. Comprehensive Graph Gradual Pruning

The proposed CGP architecture is given in Fig. 2. The CGP can obtain well-performing sparse graphs ($A'$ and $X'$) and GNN model weights ($W'$) in one single training process. In this section, we first introduce the comprehensive sparsification technique, which cosparsifies all the three elements found...
in GNN methods. Next, we describe the gradual magnitude pruning scheme, which performs the sparsification before convergence in the training stage. Finally, a regrowth scheme is given, which helps recover the pruned critical connections to refine the pruning process.

1) Comprehensive Graph Sparsification: According to (1), there are three elements, $A$, $X$, and $W$, which can impact the training and inference cost. To this end, our method simultaneously reduces the edges ($A$) and the feature dimension ($X$) in the graph $G$ and the model weights ($W$) in GNNs, whereas previous LTH-based methods only focus on cosparsifying $A$ and $W$.

1) We begin with the model weight pruning ($W$). Specifically, during the model initialization stage, we create a non-differentiable binary mask $m_w$, which is of the same size as the model weights, $W$. The elements in the mask are initially set to unity. At regular intervals (described in the next paragraph), the mask matrix is updated by our pruning strategy that sets the parameters below the threshold to zero, and the weight matrix is multiplied with the updated mask to determine which of the weights participate in the next forward execution of the graph. This procedure can be described as

$$
\text{idx} = \text{TopK}(-|W|, [p_w||W||0])
$$

$$
\begin{align*}
m_w' &= \text{Zero}(m_w, \text{idx}) \\
W' &= m_w' \odot W
\end{align*}
$$

where $\text{TopK}$ is the function that returns the indices of the top $[p_w||W||0]$ values in $|W|$, $\text{Zero}$ is the function that sets the values in $m_w$ with indices $\text{idx}$ to 0, $W'$ is the pruned weight matrix, $p_w$ is the sparsity of model weights (i.e., retaining $1 - p_w$ proportion of weights for the next iteration), $||W||0$ is the number of model weights, $\lceil \cdot \rceil$ is the rounding up operator, and $\odot$ is the elementwise product. Note that we adopt the global pruning by default, which refers to pruning the weights of different layers together.

2) Then, we introduce the graph structure pruning ($A$). Different from the model weights, which are updated at every optimizer update step, the adjacency matrix is usually fixed along the training progress. Therefore, we introduce an adjacency attention mask, $m_a \in \mathbb{R}^{||A||0}$, a sparse and differentiable soft-mask, where $||A||0$ is the number of the edges that we consider for pruning. After every optimizer update step, the soft-mask, $m_a$, is sent to the GNN model, serving as the “edge weight,” which is similar to the attention values in GAT [3]. Note that soft mask indicates that the value of mask matrix is not only 0 or 1 but also arbitrary decimals, such as 0.5 or 0.6. The proposed soft mask can help solve the problem of slackness in hard filter pruning. Denoting the value of the adjacency matrix, edge weight can be any value including 1, which represents the strength of the relationship between the two nodes. Edge weight is commonly used in PyG [52]. At regular intervals, usually in sync with the model weight pruning interval, these masks are updated by setting all the parameters lower than the threshold to zero, that is,

$$
\text{idx} = \text{TopK}(-|m_a|, [p_a||A||0])
$$

$$
\begin{align*}
m_a' &= \text{Zero}(m_a, \text{idx}) \\
A'_\text{nonzero} &= m_a' \odot A_{\text{nonzero}}
\end{align*}
$$

where $p_a$ is the graph structure sparsity, $A_{\text{nonzero}}$ denotes the edge index. $A'_\text{nonzero}$ is the pruned edge index used in the next training iteration, and $\text{idix}$ is an indexing operation.

3) Finally, we use the node feature pruning ($X$). Besides the graph structure and model weight pruning strategies, we additionally prune the dimension of node features, which is commonly used in PyG [52].
features, which is another basic element in all the GNNs. Removing the input features corresponds to removing rows or columns in the weight matrices while sparsifying weights removes the elements of these matrices. In addition, node features typically have large amounts of redundancy for the downstream tasks (e.g., node classification) [53]. Therefore, like graph structure pruning, we introduce a feature attention mask, \( m_s \in \mathbb{R}^{d \times 1} \), a sparse and differentiable soft-mask, where \( d \) is the dimension of node features. This soft mask is applied to the input layer to guide input node feature pruning. Therefore, the formulation is as follows:

\[
\begin{align*}
\text{idx} &= \text{TopK}(-|m_s|, [p_s, d]) \\
\hat{m}_s &= \text{Zero}(m_s, \text{idx}) \\
M &= (\hat{m}_1, \hat{m}_2', \ldots, \hat{m}_n')^T \\
X' &= M \odot X
\end{align*}
\]

where \( p_s \) denotes the sparsity of the feature dimension, \( \hat{m}'_1 = \hat{m}'_2 = \ldots = \hat{m}'_n = \hat{m}'_s \), and \( X' \) is the pruned feature matrix used in the next training iteration. Note that in this work we only prune the input features and leave channel pruning (i.e., including embedding pruning) for future work.

2) Gradual Magnitude Pruning: The sparsity of candidate elements, introduced in the above paragraph, is often trained with a schedule. According to the previous works [24], [54], there mainly exist three different classes of training schedules: train-then-sparsify, before-training-sparsity, and sparsify-during-training.

1) The train-then-sparsify schedule performs sparsification after a standard dense training procedure, that is, it runs until convergence in \( T \) iterations, followed by a retraining procedure. The schedule usually needs to be performed over many cycles and therefore suffers from an extremely large computation cost. Furthermore, training a dense model to converge may cause overfitting, which is hard to correct with the subsequent pruning operation [24].

2) The before-training-sparsity schedule performs sparsification before the main training process, which usually suffers from inadequate performance.

3) The sparsify-during-training schedule performs sparsification before the models have been trained to converge \((T)\), which can simultaneously exhibit the training/inference efficiency and comparable performance.

The LTH-based GNN sparsification methods [19], [21], [22] are trained with the train-then-sparsify schedule, resulting in a huge training computation cost. On the contrary, we adopt the sparsify-during-training schedule, and for the first time generalize it to GNNs. More specifically, we gradually prune not only the model weights like in convolutional DNNs but also the GNN-specific elements, including the graph structure and node features. The gradual magnitude pruning [54], [55], [56] prunes the elements over \( n \) pruning iterations to reach the desired sparsity, as shown in Fig. 1. When the sparsification of all the elements is performed every \( \Delta t \) steps, then the pruning rate of each pruning iteration becomes

\[
p_t = p_f + (p_i - p_f)
\left(1 - \frac{t - t_0}{n \Delta t}\right)^3
\]

\[
t \in \{t_0, t_0 + \Delta t, \ldots, t_0 + n \Delta t\}
\]

where \( p_i \) is the initial sparsity degree, \( p_f \) is the target sparsity, \( t_0 \) is the starting epoch of gradual pruning, and \( p_i \) refers to \( p_u, p_a, \) or \( p_s \). The above gradual pruning scheme contributes to a rapid pruning in the initial phase when the redundant connections are abundant and a gradual reduction of connections that are pruned each time as there are fewer and fewer connections remaining. After obtaining the pruning rate, we prune the elements with the smallest magnitude as presented in (2)–(4), as this has become a standard method for pruning during the training stage.

3) Regrowth: Premature pruning may occur during the pruning progress, especially in the early iterations, causing the loss of important information. To correct the “mistaken” pruning, we may incorporate the regrowth schemes, including random regrowth [57], [58], gradient-based regrowth [59], and momentum-based regrowth [60], into the gradual pruning schedule. To ensure that the model remains of approximately the same size, we first remove a proportion of elements before performing regrowth [see Fig. 2(c) (bottom)].

Consider regrowing the edges in the graph \((A)\) with the regrowth scheme based on gradient [59] as an example. To this end, we first identify the unimportant connections as those with the smallest magnitude (edge weights \( m_e \)), since small magnitude indicates that either the connection’s gradient is small or a large number of oscillations affected the gradient direction. Therefore, these edges have a small contribution to the decrease in training loss and can be removed. Assuming that the regrowth ratio is \( r \), we first remove the \( r \)th proportion of unimportant edges [see Fig. 2(c) (left)] with the smallest magnitude as

\[
\text{idx} = \text{TopK}(-|m_e|, r)
\]

\[
A'_\text{nonzero} = \text{Zero}(A'_{\text{nonzero}}, \text{idx})
\]

where \( A'_{\text{nonzero}} \) is the edge matrix after removing the \( r \)th proportion of edges. Next, we regenerate the \( r \)th proportion of new connections [see Fig. 2(c) (right)] based on the gradient magnitude, given by

\[
\text{idx} = \text{TopK}(|g_{\not\in A'_{\text{nonzero}}}|, r)
\]

\[
A''_{\text{nonzero}} = A'_{\text{nonzero}} + \text{Zero}(A''_{\text{nonzero}}, \text{idx})
\]

where \( |g_{\not\in A'_{\text{nonzero}}}| \) is the gradient/momentum magnitude of the removed edges, and \( A''_{\text{nonzero}} \) is the edge matrix after the whole regrowth process. The regrowth of the other two elements, \( W \) and \( X \), is the same as that of \( A \). The above regrowth operation is performed every \( \Delta t \) intervals from the beginning of the training to the end of the pruning. Note that the regrowth scheme is much more training efficient according to the analysis in the previous works [54], [59]. Specifically, we perform regrowth immediately after each gradual pruning step, so that the regrowth occurs only once several iterations. The CGP uses the dense gradient to identify the important connections, only when performing the regrowth operation.
In the rest of the training time, the backward of CGP is sparse. Therefore, the extra overhead for calculating the regrowth is lower compared with the whole training costs.

4) Overall Architecture: We now describe the full structure of CGP, consisting of gradual pruning and regrowth (see Algorithm 1). Given a graph $G = (A, X)$, and a GNN model with initialized weights, $W$, the training [see (1)] can be compactly represented as

$$Z = f((A, X), W).$$

During the training stage, at every $\Delta t$ training steps, we apply the pruning once on $A$, $X$, and $W$, respectively. The pruning rate is calculated by (5), and the specific pruning operation is presented in (2)–(4). After the pruning operation, we also apply the regrowth operation to “correct” the pruning error by (6) and (7) before the next round of training.

5) Complexity Analysis: We consider GCN [1] as an example. The time and memory complexities of the basic GCN at the inference stage are $O(L|A|d^2) + Lnd^2$ and $O(Lnd + Ld^2)$, respectively, where $L$, $n$, $|A|$, and $d$ are, respectively, the total number of GCN layers, nodes, edges, and the dimension of features. For the final sparsities of $W$, $A$, and $X$ as $p_w$, $p_a$, and $p_x$, then the inference time and memory complexities of GCN resulting from our proposed CGP are, respectively, $O(L(1 - p_w)|A|d + (1 - p_w)Ln(1 - p_x)d^2)$ and $O(Ln(1 - p_a)d + (1 - p_a)L((1 - p_x)d^2$).

6) Analysis on Gradient Flow: Gradient flow is a useful tool to study the optimization dynamics of sparse networks [61], [62], [63], [64]. Mathematically, the gradient of a network is defined as $g = (\partial L/\partial \theta)$, where $L$ is the loss function and $\theta$ is the vector of network weights. Then, the gradient flow $g_f$ is typically computed as

$$g_f = \|g\|_q$$

where $\|\cdot\|_q$ denotes the $q$th norm, and traditional gradient flow measures usually use the $L_1$ norm [62], [64]. According to the analysis in [61], a larger gradient norm indicates that each gradient update achieves a greater loss reduction. Therefore, we desire to preserve or even increase the gradient norm.

Fig. 3 shows the gradient norm of pruned networks on the Cora and Photo datasets. The gradient norm is obtained by applying the $L_1$ norm to a single vector of all the gradients concatenated together. From the results, we can observe that: 1) our CGP yields higher gradient norm early in training than base model (GCN) and other SOTA sparsifying methods on the Cora dataset, which mitigates the vanishing gradient problem and provides more training progress [65], [66] and 2) CGP provides a more stable gradient norm at the end of the training process, compared with ICPG on the Photo dataset. In summary, our CGP helps generate healthy gradient flow during the training of sparse neural networks, which partly explains its success in pruning GNNs (see Section IV).

C. Further Discussion

1) CGP Versus LTH-Based GNN Sparsification: To the best of our knowledge, the methods most relevant to our CGP are the LTH-based GNN sparsification methods, including UGS [21], GEBT [19], and ICPG [22]. The original motivation for these methods is to trim down the inference Multiplication-and-ACCumulation operations (MACs) without sacrificing performance. More specifically, the LTH-based methods attempt to identify the GLT, consisting of a sub-network ($W_{\text{sparse}} \subset W$) and a sparse graph ($A_{\text{sparse}} \subset A$), which maintains the performance of dense networks on the full graph. To achieve the above goal, they usually adopt an iterative training cycle [see Fig. 1 (left)]: 1) train-dense; 2) prune; and 3) retrain, until the desired sparse rates for the graph structures and model weights are, respectively, reached. While the LTH-based methods trim down the inference MACs, they suffer from a huge training cost since they require many pruning and retraining cycles to achieve the desired performance (up to $20 \times$). Our CGP method, however, can obtain well-performing sparse graphs ($A'$ and $X'$) and GNN model weights ($W'$) with one single training process [see Fig. 1 (right)]. Therefore, the CGP exhibits training and inference efficiency compared with dense GNNs while maintaining comparable performance of GNN models.

2) CGP Versus Graph Sampling: The goal of graph sampling or neighborhood selection is to extract a small subgraph from the original graph, which remains effective for learning tasks while also reducing the costs. GraphSAGE [2] is a nodewise sampling method, which randomly selects two-hop neighbors for each node. The aggregation process is performed based on the sampled nodes. FastGCN [67] introduces a global neighbor sampling method instead of a local one like GraphSAGE. RioGNN [68] proposes a relation-aware neighbor selection mechanism to choose the most similar neighbors to a target node in the relational graph. search to aggregate neighborhood (SANE) [69] automatically designs

Algorithm 1 Pseudocode for the Proposed CGP

Require: Graph $G = (A, X)$, GNN $f(G, W)$, GNN initialization, $W$, initial masks, $m_w$, $m_a$, and $m_x$, target sparsity, $p_w$, $p_a$, and $p_x$, gradual pruning starting point, $t_0$, gradual pruning endpoint, $t_f$, gradual pruning frequency, $\Delta t$.
1: for each training step $t$ do
2: Forward $f(\cdot, m_w \odot W)$ with $G = (m_w \odot A, m_a \odot X)$
3: Backpropagate to update $W$, $m_w$, and $m_x$
4: if $t_0 \leq t \leq t_f$ and $(t \mod \Delta t) == 0$ then
5: Gradual pruning $W$, $A$, and $X$ by (2), (3), and (4), respectively, with the pruning rate produced by (5)
6: Regrow $W$, $A$, and $X$ with (6) and (7)
7: end if
8: end for

Authorized licensed use limited to the terms of the applicable license agreement with IEEE. Restrictions apply.
TABLE I
STATISTICS AND PROPERTIES OF BENCHMARK DATASETS

| Dataset       | # Nodes | # Edges | # Yea. | # Cla. | Homo. | Degree |
|---------------|---------|---------|--------|--------|-------|--------|
| Heterophily Datasets |
| Cornell       | 183     | 280     | 1703   | 5      | 0.30  | 1.64   |
| Texas         | 183     | 195     | 1703   | 5      | 0.11  | 1.77   |
| Wisconsin     | 251     | 466     | 1703   | 5      | 0.21  | 2.05   |
| Actor         | 7,600   | 26,752  | 932    | 5      | 0.22  | 3.94   |
| Small-scale Homophily Datasets |
| Cora          | 2,708   | 5,429   | 1,433  | 7      | 0.83  | 3.90   |
| CiteSeer      | 3,327   | 4,732   | 3,703  | 6      | 0.72  | 2.78   |
| PubMed        | 19,717  | 44,338  | 500    | 3      | 0.79  | 4.50   |
| Medium-scale Homophily Datasets |
| Physics       | 34,493  | 247,962 | 500    | 5      | 0.91  | 14.38  |
| CS            | 18,333  | 81,894  | 500    | 15     | 0.83  | 8.93   |
| Photo         | 7,650   | 119,081 | 745    | 8      | 0.85  | 31.13  |
| Computers     | 13,752  | 245,861 | 767    | 10     | 0.81  | 35.76  |
| Large-scale Homophily Datasets (OGBN) |
| ogbn-arxiv    | 169,343 | 1,166,343 | 128   | 40     | 0.63  | 7.68   |
| ogbn-proteins | 132,534 | 39,561,252 | 8    | 2      | –     | 597.00 |
| ogbn-products | 2,449,029 | 61,859,140 | 100  | 47     | 0.81  | 50.51  |

data-specific GNN architectures with a differentiable search algorithm. In summary, all the above methods tend to reduce the costs from a single point of view of graph structure, while our method CGP comprehensively exploits the three core elements of GNNs, namely, graph structures, node features, and model parameters, to improve the training and inference efficiency.

IV. EXPERIMENTS

To validate the effectiveness of the proposed method, we conducted extensive experiments across six GNN architectures on 14 benchmark graph datasets. Eleven observations emerged from the experiments.

A. Experimental Settings

1) Datasets: We used a total of 14 real-world datasets, including ten homophilic graph datasets (Cora, Citeseer, Pubmed, CS, Physics, Photo, Computers, ogbn-arxiv, ogbn-proteins, and ogbn-products) and four heterophilic graph datasets (Cornell, Texas, Wisconsin, and Actor), involving diverse domains (citation, coauthorship, copurchase, and web pages) and sizes (ogbn-arxiv, ogbn-proteins, and ogbn-products are large-scale datasets from the OGB [13]). All the above datasets were used at their original settings (e.g., splitting) whenever possible. The dataset statistics are summarized in Table I. Note that Homo. is defined as the proportion of edges connecting nodes with the same label [70].

2) Models: The models are described as follows.

1) Six Baseline Models: We selected six representative GNNs as backbone models, namely, GCN [1], GAT [3], SGC [25], APPNP [26], ResGCN [14], and GCNII [16]. Note that ResGCN and GCNII are two deeper graph neural models, over which we conducted experiments with large-scale datasets such as ogbn-arxiv, ogbn-proteins, and ogbn-products. Our CGP can be applied to these methods to further improve their performance.

2) Three SOTA Sparsification Models for GNNs: To comprehensively evaluate the CGP model, we compared it with three state-of-the-art sparsification methods for GNNs, namely, UGS [21], which extends LTH to GNNs and proposes the GLT, GEBT [19], which extends the Early-Bird Tickets Hypothesis [45] to GNNs, and ICPG [22], which endows the GLT with the inductive learning.

3) Six SOTA Heterophilous-Oriented Models: To further evaluate the effectiveness of CGP on heterophilous graphs, we compared it with several benchmarking heterophilous-oriented models, including multilayer perceptron (MLP) [71], which only uses node features, logistic regression on the adjacency matrix (LINK) (logistic regression on the adjacency matrix) [72], which only uses the graph topology, and GCN with homophily or heterophily (H2GCN) [70], Mix-hop [73], generalized pagerank GNN (GPRGNN) [74], and frequency adaptation graph convolutional networks (FAGCN) [75], which are designed for heterophilous graphs.

B. Shallow GNNs on Small- and Medium-Scale Datasets

1) Proposed CGP Versus Baseline Models: To evaluate the benefits of our proposed CGP, we first compared the performance of CGP over four commonly used GNN models on ten benchmark graph datasets from three perspectives: 1) cosparsifying model weights and graph structures (W&A); 2) cosparsifying model weights and node features (W&X); and 3) cosparsifying model weights, graph structure, and node features (W&A&X). The results are shown in Figs. 4–6, respectively. Note that when sparsifying W&A&W&X, we keep X/A complete. From the results, we obtain the following observations.

Obs.1 (The Extreme Sparsity of Model Weights Can Still Maintain Satisfactory Performance): As shown in Fig. 4, the performance of 99.9% weight sparsity (i.e., only 0.1% of model weights remain at the inference stage) is consistently comparable to that of low-level weight sparsity (e.g., 99.0%, 90.0%, and 50.0%) over different graph sparsities on most models and datasets. Furthermore, on several small-scale datasets (e.g., Cornell and Actor), the extreme sparsity of model weights can maintain higher accuracies compared with other sparsities. Some observations apply to Figs. 5 and 6.

Obs.2 (Denser Graphs Are More Resilient to Graph Sparsification): As shown in Fig. 4, the performance on the Computers and Photo datasets does not decrease as the sparsity becomes larger, in most cases. This can be attributed to the

---

3 Codes are released on https://github.com/LiuChuang0059/CGP.
Fig. 4. Results of cosparsifying model weights and graph structures (W&A), with the proposed CGP compared with backbone models (GCN and APPNP) over six graph datasets. From left to right: Two small-scale, high-homophily, and small node degree datasets (Citeseer and Pubmed); two middle-scale, high-homophily, and large node degree datasets (Computers and Photo); and two small-scale and high-heterophily datasets (Cornell and Actor). The results are summarized in Obs.1, Obs.2, Obs.3, and Obs.5.

Fig. 5. Results of cosparsifying model weights and node features (W&X), when evaluating GCN on two homophilic graphs (Citeseer and Pubmed) and one heterophilic graph (Actor). The feature dimensions of datasets from left to right: 3703, 500, and 932. Baseline denotes the performance of vanilla GCN on the complete graphs. The maximum feature sparsity reaches 99.0%, which means only 1% of feature channels remain at the reference stage. The results are summarized in Obs.4.

3703 features. As shown in Fig. 5, the model usually achieves the best performance when the feature sparsity is about 70% or 80%, which indicates that the model does not require too many features for each node when performing inference. Therefore, by aggressively trimming down the node features, performance can still be maintained or even be improved.

Obs.3 (CGP Achieves a Great Improvement on Heterophilic Graphs): In a heterophilic (or low homophily) graph (e.g., Cornell), the connected nodes may have different class labels and dissimilar features with a high probability. Therefore, most edges can be seen as “noisy” edges for the node classification task and should be cut off. As shown in Fig. 4, the model performance becomes better as the graph sparsity becomes higher (i.e., cutting off more edges), which supports our above analysis. Note that different from MLP, which does not use the edges from beginning to end, our method gradually reduces the number of edges instead of cutting off all the edges. This enables our method to take full advantage of the “true” edges. We also present a detailed comparison of MLP and our CGP in the next part (see Table II).

Obs.4 (The Node Feature Channels in Most Graphs Are Redundant): The dimension of node feature vectors in graphs can be very high, e.g., each node in the Citeseer graph has 3703 features. As shown in Fig. 5, the model usually achieves the best performance when the feature sparsity is about 70% or 80%, which indicates that the model does not require too many features for each node when performing inference. Therefore, by aggressively trimming down the node features, performance can still be maintained or even be improved.

Obs.5 (Graph Structure Is More Sensitive to Pruning): As shown in Fig. 6, when we cosparsify W&A&X, the graph structure, A, always plays a more important role in the model performance. Specifically, the gap between the performance of different graph sparsities is larger than that of different feature sparsities and weight sparsities on Pubmed and Wisconsin. The former is a homophilic graph, and the latter is a heterophilic one. The same observation can also be obtained when we cosparsify W&A, as shown in Fig. 4.

2) Proposed CGP Versus SOTA Sparsification Methods for GNNs: To further explore the effectiveness of the proposed CGP, we compared CGP with several SOTA LTH-based GNN sparsification pipelines, including UGS [21], GEBT [19], and ICPG [22], in terms of Accuracy (see Fig. 7), FLOPs (see Fig. 8), and Training Time (see Fig. 9). For a fair comparison, we adopted two different settings, one-shot pruning and iterative pruning, for the UGS and ICPG methods. Note that our CGP and GEBT methods always adopt one-shot pruning.
which is much more effective than iterative pruning during the training process. We summarize the results in the following observations.

**Obs.6 (CGP Consistently Outperforms SOTA LTH-Based Methods):** As shown in Fig. 7, all the methods adopt one-shot pruning, using only one pruning step to reach the final desired sparsity. We can observe that CGP consistently surpasses the SOTA LTH-based methods by substantial performance margins across all the datasets and GNNs, which validates the effectiveness of our proposed framework. In Fig. 8, UGS and ICPG adopt iterative pruning, which requires a comprehensive and iterative (up to 20×) training of dense models to reach the desired sparsity, while our CGP and GEBT methods continue to adopt one-shot pruning. Therefore, our CGP method requires the smallest number of training FLOPs while achieving a comparable performance across all the datasets.

**Obs.7 (CGP Is More Amenable to the Extreme Sparsity Than SOTA LTH-Based Methods):** In Figs. 7 and 8, the performance of our method does not degrade significantly with the increase in sparsity rates (the decrease in the inference FLOPs) compared with the SOTA methods, which suggests that CGP is more amenable to the extreme sparsity. Especially when the weight sparsity reaches 99.9% (only 0.1% edges of graph remain at the inference stage), UGS and ICPG completely fail to handle this case, limiting their applications in the resource-constrained situations (e.g., mobile phones).

**Obs.8 (CGP Substantially Saves Inference MACs):** As shown in Fig. 8, when the inference FLOPs (1 FLOPs ≈ 2 MACs) reaches about only 100 M, up to 99% MACs saving of the baseline models, CGP can still provide satisfactory performance and maintain higher accuracies on severely sparsified graphs.

**Obs.9 (CGP Better Boosts Training Efficiency Compared With SOTA LTH-Based Methods):** Fig. 9 shows the performance of the GNN sparsification methods over their training time relative to that of base GCN on the Computers and Photo datasets. For a fair comparison, we set all the methods with the same sparsification rate and objects (A and W). We observe that the training time of UGS is about 45× bigger than that of the baseline model (GCN), and that our method can be trained up to about two orders of magnitude faster than the LTH-based methods while maintaining higher accuracies.

3) **Proposed CGP Versus SOTA Heterophilous-Oriented Methods:** Furthermore, we compared the CGP with several SOTA GNN methods, which were recently proposed for nonhomophilous settings on heterophilic graphs (H2GCN [70], MixHop [73], GPRGNN [74], and FAGCN [75]). From the results in Table II, we obtain the following observation.
TABLE II
OUR CGP VERSUS SOTA HETEROPHILOS GNN METHODS, **BOLD:**
THE BEST PERFORMANCE PER BENCHMARK. **UNDERLINE:** THE
SECOND BEST PERFORMANCE PER BENCHMARK

|                      | Cornell | Texas | Wisconsin | Actor |
|----------------------|---------|-------|-----------|-------|
| GCN [1]              | 59.46   | 62.16 | 54.90     | 30.39 |
| SGC [25]             | 56.76   | 59.46 | 54.90     | 26.84 |
| APPNP [26]           | 40.54   | 59.46 | 66.67     | 27.30 |
| LINK [72]            | 59.16   | 78.38 | 66.67     | 22.83 |
| MLP [71]             | 78.38   | 78.38 | 74.51     | 35.13 |
| H2GCN [70]           | 78.38   | 81.08 | 84.31     | 35.66 |
| MixHop [73]          | 56.76   | 64.86 | 66.67     | 27.57 |
| GPRGNN [74]          | 83.78   | 78.38 | **88.24** | 36.38 |
| FAGCN [75]           | 81.08   | 72.97 | 72.35     | 26.33 |

Our (GCN) vs. GCN **89.19 (150.0%)** **89.19 (143.4%)** **88.24 (160.7%)** **39.21 (129.4%)**
Our (SGC) vs. SGC **56.76 (0.0%)** **67.57 (113.6%)** **58.82 (77.1%)** **27.70 (73.2%)**
Our (APPNP) vs. APPNP **89.19 (120.0%)** **91.89 (54.5%)** **88.24 (124.4%)** **38.82 (42.2%)**

Fig. 10. Results of sparsifying two deeper GNN models: GCNII on three small- and medium-scale datasets (Pubmed, CS, and Physics) and ResGCN (28-layer) on three large-scale datasets (ogbn-arxiv, ogbn-products, and ogbn-proteins). The results are summarized in Obs.11.

Obs.10 (CGP Consistently Performs Better Than Hetero-GNN Methods): We conclude the results on the heterophilic graph datasets in Table II, where ▲ denotes an improvement over the original baseline models. It can be seen that CGP consistently outperforms all the baselines and SOTA methods in terms of accuracy. Specifically, CGP achieves up to 120% improvements when compared with baselines (GCN [1], SGC [25], and APPNP [26]). Furthermore, compared with MLP, which cuts off all the edges, our method also consistently reaches higher accuracies on all the heterophilic graph datasets. One possible explanation is that our method gradually reduces the number of edges, which means that we can take advantage of the useful edges during the training stage. We also applied our method to the SOTA Hetero-GNN methods, such as H2GCN [70] or FAGCN [75], to further improve their efficiency and accuracy.

C. Scale to Deeper GNNs

In this section, we further conducted experiments with deeper GNNs (ResGCN [14] and GCNII [16]), which induce a large amount of weights (W) on small- and large-scale datasets. Specifically, we tested GCNII (64-layer) on three small-and medium-scale datasets (Pubmed, CS, and Physics), and ResGCN (28-layer) on three large-scale datasets (ogbn-arxiv, ogbn-products, and ogbn-proteins), which have a large amount of edges (A). The results are illustrated in Fig. 10 and summarized in the following observation.

Obs.11. (CGP Is Scalable for Deep GNNs): Fig. 10 demonstrates that CGP can be scaled up to deep GNNs. First, CGP achieves matching performance with base GCNII with about 95% and 90% MACs saving on the CS and Physics datasets, respectively. Also, CGP almost outperforms base GCNII on Pubmed (up to 98% MACs saving). Second, CGP obtains matching performance with base ResGCN with 50% and 25% MACs saving on ogbn-arxiv and ogbn-proteins, respectively. Furthermore, our CGP consistently outperforms base ResGCN on ogbn-products (up to 80% MACs saving).

D. Ablation Study

1) Sparsifying W, A, X Separately: In this section, we separated the sparsifying operations and explored their roles when applying them to the graphs and the model independently. From Fig. 11, we can state the following observations. First, sparsifying the model weights usually helps improve the model performance. Second, in most graphs, the trends of different sparsifying ways are inconsistent, which requires us to find a balance point. More specifically, structure pruning outperforms other strategies on the Computer and Actor datasets, while weight pruning and feature pruning perform relatively better on Citeseer and Pubmed, respectively.

2) Effectiveness of Regrowth and Pruning Ways: In this section, we investigated the influence of different regrowth ways, including random growth [57], [58], momentum-based growth [60], gradient-based growth [59], and no growth. From Fig. 12 (left), we can observe that all the regrowth mechanisms help boost the performance, especially when the model becomes extremely sparse. Also, the momentum-based regrowth usually performs better than other methods. In addition, we also explored the rules of the pruning way, including layerwise pruning and global pruning. More specifically, global pruning prunes different layers together and leads to nonuniform sparsity distributions, while layerwise pruning performs the operation layer by layer, resulting in uniform distributions. From Fig. 12 (right), we observe that as the model weights become sparser, the global pruning way performs better than the layerwise pruning. This may be because...
the global pruning can preserve more connections on important layers.

E. Further Study

1) Sparse-to-Sparse Training Schema: Different from the LTH-based methods, which only perform pruning on dense networks and full graphs, our proposed CGP can also support a sparse-to-sparse training schema, which means that our method can be trained with sparse initialization weights and graphs. In this section, we take sparsifying weights as an example and illustrate the results in Fig. 13. Note that except for the initial density of model weights, other experimental settings are the same as those in the above experiments. Somewhat surprisingly but fully justifiable, it seems that the smaller the initial sparsity (up to 50%), the higher the test accuracy of CGP. In particular, on the Cora and Citeseer datasets, 50% weight sparsity initialization usually outperforms the dense initialization, which provides a possibility to train large or deep GNN models on smaller capacity devices. This is probably because it is hard to train much more parameters of GNNs on small-scale datasets. We leave the training with a sparse initial graph structure and node features for future work.

2) Feature Pruning Versus Traditional Feature Engineering: Models: We selected five representative feature engineering methods.

1) Principal component analysis (PCA).
2) Truncated singular value decomposition (SVD).
3) Nonnegative Matrix Factorization (NMF): Decomposing signals with matrix factorization [76], [77], [78].
4) T-Distributed Stochastic Neighbor Embedding (t-SNE): It converts similarities between data points to joint probabilities and tries to minimize the Kullback–Leibler divergence between the joint probabilities of the low-dimensional embedding and the high-dimensional data [79].
5) SpectralEmbedding (SE): Spectral embedding for non-linear dimensionality reduction [80].

Datasets: We selected two datasets, one homophilic dataset (Citeseer) and one heterophilic dataset (Cornell), with accuracy for evaluation metric. The feature dimensions of these two datasets are relatively larger than those of other datasets.

Implementation Details: For a fair comparison, we fixed all the parts of vanilla model (GCN) except for the node feature processing. For all the five methods except t-SNE, we tuned the number of components from \{2, 10, 20, 50, 100, 200\}. And for t-SNE, we fixed the number of components as 2. For our CGP, we only adopted the feature pruning strategy.

Experimental Results: The results are plotted in Fig. 14, from which we can observe that: 1) all the five traditional feature engineering methods decrease the performance of the baseline models on the two datasets and 2) our CGP consistently outperforms all the traditional feature engineering methods and achieves competitive results compared with vanilla models (Base in Fig. 14). This is likely because in real-world applications, node attributes in a graph usually contain irrelevant and redundant noises which may affect the effectiveness and performance of GNNs’ learning [81], [82], [83]. Our CGP with feature pruning acts as a feature selection module which helps improve many learning performances. Moreover, our feature selection is a “soft-selector,” which assigns different learnable importance to the feature dimension according to the back-forward gradients.

3) Analysis of GPU and CPU Memory Efficiency: We compared our method with the baseline method (GCN) and three SOTA sparsification methods for GNNs, namely, UGS, GEBT, and ICPG, on three benchmark datasets (Cornell, Cora, and Photo). The results are shown in Fig. 15, from which we can observe that: 1) CGP is more efficient than all the three SOTA sparsification methods in terms of GPU memory and GPU memory and 2) CGP consumes on par or more of GPU memory, and on par or less of CPU memory, compared with the baseline method. It is obvious that it is challenging for all the unstructured sparsity methods, including UGS, GEBT, ICPG, and our CGP, to accelerate GNNs on commercially available GPUs, since current GPUs are well-optimized for dense matrix multiplications but limited in sparse matrix multiplications.

However, unstructured sparsity methods are more promising and capable of matching the dense models at very high sparsity levels, compared with structured sparsity [57]. If hardware and software engineering challenges in sparse matrix multiplications are solved, our CGP may prove a basis for deeper and larger GNNs and lead to the building of efficient and effective GNNs, which could be directly trained on resource-constrained devices (e.g., mobile phones), without the need to first train them on supercomputers and then to deploy the trained models on the resource-constrained devices. Moreover, our CGP can help save a large amount of energy and reduce carbon emissions on the resource-constrained devices. For all the five methods except t-SNE, we tuned the number of components from \{2, 10, 20, 50, 100, 200\}. And for t-SNE, we fixed the number of components as 2. For our CGP, we only adopted the feature pruning strategy.

The results are plotted in Fig. 14, from which we can observe that: 1) all the five traditional feature engineering methods decrease the performance of the baseline models on the two datasets and 2) our CGP consistently outperforms all the traditional feature engineering methods and achieves competitive results compared with vanilla models (Base in Fig. 14). This is likely because in real-world applications, node attributes in a graph usually contain irrelevant and redundant noises which may affect the effectiveness and performance of GNNs’ learning [81], [82], [83]. Our CGP with feature pruning acts as a feature selection module which helps improve many learning performances. Moreover, our feature selection is a “soft-selector,” which assigns different learnable importance to the feature dimension according to the back-forward gradients.

3) Analysis of GPU and CPU Memory Efficiency: We compared our method with the baseline method (GCN) and three SOTA sparsification methods for GNNs, namely, UGS, GEBT, and ICPG, on three benchmark datasets (Cornell, Cora, and Photo). The results are shown in Fig. 15, from which we can observe that: 1) CGP is more efficient than all the three SOTA sparsification methods in terms of GPU memory and GPU memory and 2) CGP consumes on par or more of GPU memory, and on par or less of CPU memory, compared with the baseline method. It is obvious that it is challenging for all the unstructured sparsity methods, including UGS, GEBT, ICPG, and our CGP, to accelerate GNNs on commercially available GPUs, since current GPUs are well-optimized for dense matrix multiplications but limited in sparse matrix multiplications.

However, unstructured sparsity methods are more promising and capable of matching the dense models at very high sparsity levels, compared with structured sparsity [57]. If hardware and software engineering challenges in sparse matrix multiplications are solved, our CGP may prove a basis for deeper and larger GNNs and lead to the building of efficient and effective GNNs, which could be directly trained on resource-constrained devices (e.g., mobile phones), without the need to first train them on supercomputers and then to deploy the trained models on the resource-constrained devices. Moreover, our CGP can help save a large amount of energy and reduce carbon emissions on the resource-constrained devices.
emissions caused by machine learning. Fortunately, there is a growing number of works for speeding-up sparse neural networks on the practical hardware [84], [85], [86], which provides direction for translating the efficiency of unstructured sparsity into real speedups.

Currently, therefore, the computational efficiency of unstructured pruning methods is mainly evaluated by FLOPs. All the three SOTA graph sparse training methods, namely, UGS, ICPG, and GEBT, use FLOPs as the evaluation index of computational efficiency, as shown in Figs. 8 and 10.

V. CONCLUSION

To boost both training and inference efficiency, we have proposed an effective during-training graph pruning (CGP) framework to dynamically prune GNNs from three aspects (A, W, and X) at the training stage and within one single training process, which helps greatly save the computational cost and achieve comparable or even better performance compared with the LTH-based methods. We have evaluated the proposed method over the node classification task across six (various) GNN architectures, on 14 real-world graph datasets. The experimental results have demonstrated that our proposed method exhibits both higher training and inference efficiency while offering comparable or even higher accuracy when compared with the state-of-the-art methods. Furthermore, our method contributes to the reduction of both energy costs and CO2 emissions caused by the training of machine learning models.

In future work, first, we will further investigate channel pruning, including the input node features pruning and the hidden embeddings’ pruning. Also, second, we are planning to design some effective sparse graph structures and node feature initializations, which will guarantee the training of large or deep models on large-scale graph datasets with a small-sized memory CPU/GPU. Furthermore, third, we will work on how to regrow the edges from the complete edge space, and not only from the existing edges. In addition, fourth, we plan to generalize our method to other graph tasks, such as graph classification and link prediction, to boost their efficiency and accuracy.

REFERENCES

[1] T. N. Kipf and M. Welling, “Semi-supervised classification with graph convolutional networks,” in Proc. 5th Int. Conf. Learn. Represent., 2017.
[2] W. Hamilton, Z. Ying, and J. Leskovec, “Inductive representation learning on large graphs,” in Proc. Adv. Neural Inf. Process. Syst., vol. 30, 2017.
[3] P. Velicković, G. Cucurull, A. Casanova, A. Romero, P. Liò, and Y. Bengio, “Graph attention networks,” in Proc. 6th Int. Conf. Learn. Represent., 2018, pp. 1–12.
[4] K. Xu, W. Hu, J. Leskovec, and S. Jegelka, “How powerful are graph neural networks?” in Proc. 7th Int. Conf. Learn. Represent., 2019, pp. 1–17.
[5] C. Li et al., “Joint stance and rumor detection in hierarchical heterogeneous graph,” IEEE Trans. Neural Netw. Learn. Syst., vol. 33, no. 6, pp. 2530–2542, Jun. 2022.
[6] Y. Hei et al., “Hawk: Rapid Android malware detection through heterogeneous graph attention networks,” IEEE Trans. Neural Netw. Learn. Syst., early access, Aug. 27, 2021, doi: 10.1109/TNNLS.2021.3105617.
[7] S. Jia, S. Jiang, S. Zhang, M. Xu, and X. Jia, “Graph-in-graph convolutional network for hyperspectral image classification,” IEEE Trans. Neural Netw. Learn. Syst., early access, Jun. 20, 2022, doi: 10.1109/TNNLS.2022.3182715.
[8] M. Tiezzi, G. Ciravegna, and M. Gori, “Graph neural networks for graph drawing,” IEEE Trans. Neural Netw. Learn. Syst., early access, Jun. 28, 2022, doi: 10.1109/TNNLS.2022.3184967.
[9] Z. Wu, S. Pan, F. Chen, G. Long, C. Zhang, and P. S. Yu, “A comprehensive survey on graph neural networks,” IEEE Trans. Neural Netw. Learn. Syst., vol. 32, no. 1, pp. 4–24, Jan. 2021.
[10] L. Stanković et al., “Data analytics on graphs part I: Graphs and spectra on graphs,” Found. Trends Mach. Learn., vol. 13, no. 1, pp. 1–157, 2020.
[11] L. Stanković et al., “Data analytics on graphs part II: Signals on graphs,” Found. Trends Mach. Learn., vol. 13, nos. 2–3, pp. 158–331, 2020.
[12] L. Stanković et al., “Data analytics on graphs part III: Machine learning on graphs, from graph topology to applications,” Found. Trends Mach. Learn., vol. 13, no. 4, pp. 332–530, 2020.
[13] W. Hu et al., “Open graph benchmark: Datasets for machine learning on graphs,” 2020, arXiv:2005.00687.
[14] G. Li, M. Müller, A. Thabet, and B. Ghanem, “DeepGCNs: Can GCNs go as deep as CNNs?” in Proc. IEEE/CVF Int. Conf. Comput. Vis. (ICCV), Oct. 2019, pp. 9266–9275.
[15] G. Li et al., “DeepGCNs: Making GCNs go as deep as CNNs,” IEEE Trans. Pattern Anal. Mach. Intell., vol. 45, no. 6, pp. 6923–6939, Jun. 2023.
[16] M. Chen, Z. Wei, Z. Huang, B. Ding, and Y. Li, “Simple and deep graph convolutional networks,” in Proc. 37th Int. Conf. Mach. Learn., 2020, pp. 1725–1735.
[17] X. Liu et al., “Survey on graph neural network acceleration: An algorithmic perspective,” in Proc. 31st Int. Joint Conf. Artif. Intell., Jul. 2022, pp. 5521–5529.
[18] S. A. Tailor, J. Fernandez-Marques, and N. D. Lane, “Degree-quant: Quantization-aware training for graph neural networks,” in Proc. 9th Int. Conf. Learn. Represent., 2021, pp. 1–22.
[19] H. You, Z. Lu, Z. Zhou, Y. Fu, and Y. Lin, “Early-bird GCNs: Graph-network co-optimization towards more efficient GCN training and inference via drawing Early-Bird lottery tickets,” in Proc. 36th AAAI Conf. Artif. Intell., 2022, pp. 1–9.
[20] K. He, X. Zhang, S. Ren, and J. Sun, “Deep residual learning for image recognition,” in Proc. IEEE Conf. Comput. Vis. Pattern Recognit., Dec. 2016, pp. 770–778.
[21] T. Chen, Y. Sui, X. Chen, A. Zhang, and Z. Wang, “A unified lottery ticket hypothesis for graph neural networks,” in Proc. 38th Int. Conf. Mach. Learn., 2021, pp. 1693–1706.
[22] Y. Sui, X. Wang, T. Chen, X. He, and T.-S. Chua, “Inductive lottery ticket learning for graph neural networks,” in Proc. ICLR, 2022, pp. 1–18.
[23] J. Frankle and M. Carbin, “The lottery ticket hypothesis: Finding sparse, trainable neural networks,” in Proc. 7th Int. Conf. Learn. Represent., 2019, pp. 1–12.
[24] T. Hoefler, D. Alistarh, T. Ben-Nun, N. Dryden, and A. Peste, “Sparsity in deep learning: Pruning and growth for efficient inference and training in neural networks,” J. Mach. Learn. Res., vol. 22, pp. 1–124, Sep. 2021.
[25] F. Wu, A. Souza, T. Zhang, C. Fitzy, T. Yu, and K. Weinberger, “Simplifying graph convolutional networks,” in Proc. Int. Conf. Mach. Learn., 2019, pp. 6861–6871.
[26] J. Gasteiger, A. Bojchevski, and S. Günnemann, “Combining neural networks with personalized PageRank for classification on graphs,” in Proc. 7th Int. Conf. Learn. Represent., 2019, pp. 1–14.
[27] T. Chen et al., “Bag of tricks for training deeper graph neural networks: A comprehensive benchmark study,” IEEE Trans. Pattern Anal. Mach. Intell., vol. 45, no. 3, pp. 2769–2781, Mar. 2023.
[28] V. Sadhanala, Y.-X. Wang, and R. Tibshirani, “Graph sparsification approaches for Laplacian smoothing,” in Proc. Artif. Intell. Statist., 2016, pp. 1250–1259.
[29] D. Calandriello, A. Lazaric, I. Koutis, and M. Valko, “Improved large-scale graph learning through ridge spectral sparsification,” in Proc. 35th Int. Conf. Mach. Learn., 2018, pp. 688–697.
[30] C. Zheng et al., “Robust graph representation learning via neural sparsification,” in Proc. 37th Int. Conf. Mach. Learn., 2020, pp. 11458–11468.
[31] Y. Rong, W. Huang, T. Xu, and J. Huang, “DropEdge: Towards deep graph convolutional networks on node classification,” in Proc. 8th Int. Conf. Learn. Represent., 2020, pp. 1–17.
[32] J. Li, T. Zhang, H. Tian, S. Jin, M. Fardad, and R. Zafarani, “SGCN: A graph sparsifier based on graph convolutional networks,” in Proc. Pacific-Asia Conf. Knowl. Discov. Data Mining, 2020, pp. 275–287.

[33] S. Boyd, N. Parikh, E. Chu, B. Peleato, and J. Eckstein, “Distributed optimization and statistical learning via the alternating direction method of multipliers,” Found. Trends Mach. Learn., vol. 3, no. 1, pp. 1–122, 2011.

[34] A. Loukas and P. Vanderghynst, “Spectrally approximating large graphs with smaller graphs,” in Proc. 35th Int. Conf. Mach. Learn., 2018, pp. 3237–3246.

[35] A. Loukas, “Graph reduction with spectral and cut guarantees,” J. Mach. Learn. Res., vol. 20, no. 116, pp. 1–42, 2019.

[36] C. Cai, D. Wang, and Y. Wang, “Graph coarsening with neural networks in,” in Proc. 9th Int. Conf. Learn. Represent., 2021, pp. 1–27.

[37] Z. Huang, S. Zhang, C. Xi, T. Liu, and M. Zhou, “Scaling up graph neural networks via graph coarsening,” in Proc. 27th ACM SIGKDD Conf. Knowl. Discovery Data Mining, Aug. 2021, pp. 675–684.

[38] Y. Liu, T. Safavi, A. Dighe, and D. Koutra, “Graph summarization methods and applications: A survey,” ACM Comput. Surveys, vol. 51, no. 3, pp. 1–34, May 2019.

[39] F. Frasca, E. Rossi, D. Eynard, B. Chamberlain, M. Bronstein, and F. Monti, “SIGN: Scalable inception graph neural networks,” 2020, arXiv:2004.11198.

[40] M. Liu, H. Gao, and S. Ji, “Towards deeper graph neural networks,” in Proc. 26th ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining, 2020, pp. 338–348.

[41] A. Bojchevski et al., “Scaling graph neural networks with approximate PageRank,” in Proc. 26th ACM SIGKDD Int. Conf. Knowl. Discov. Data Mining, 2020, pp. 2464–2473.

[42] R. Li, S. Wang, F. Zhu, and J. Huang, “Adaptive graph convolutional neural networks,” in Proc. AAAI Conf. Artif. Intell., vol. 32, no. 1, Apr. 2018, pp. 1–9.

[43] J. Frankle, G. K. Dziugaite, D. Roy, and M. Carbin, “Linear mode connectivity and the lottery ticket hypothesis,” in Proc. 37th Int. Conf. Mach. Learn., 2020, pp. 3259–3269.

[44] Z. Zhang, X. Chen, T. Chen, and Z. Wang, “Efficient lottery ticket finding: Less data is more,” in Proc. 38th Int. Conf. Mach. Learn., 2021, pp. 12380–12390.

[45] H. You et al., “Drawing Early-bird tickets: Toward more efficient training of deep networks,” in Proc. 8th Int. Conf. Learn. Represent., 2020, pp. 1–13.

[46] H. Zhou, J. Lan, R. Liu, and J. Yosinski, “Deconstructing lottery tickets: Zeros, signs, and the supermask,” in Proc. Adv. Neural Inf. Process. Syst., vol. 32, 2019.

[47] Y. Ma et al., “Purity checks for lottery tickets: Does your winning ticket really win the Jackpot?” in Proc. Adv. Neural Inf. Process. Syst., vol. 34, 2021, pp. 12749–12760.

[48] X. Chen, T. Chen, Z. Zhang, and Z. Wang, “You are caught stealing my winning lottery ticket! Making a lottery ticket claim its ownership,” in Proc. Adv. Neural Inf. Process. Syst., vol. 34, 2021, pp. 1780–1791.

[49] T. Chen, Z. Zhang, S. Liu, S. Chang, and Z. Wang, “Long live the lottery: The existence of winning tickets in lifelong learning,” in Proc. 9th Int. Conf. Learn. Represent., 2021, pp. 1–19.

[50] X. Chen, Z. Zhang, Y. Sui, and T. Chen, “GANs can play lottery tickets too,” in Proc. 9th Int. Conf. Learn. Represent., 2021, pp. 1–15.

[51] T. Chen et al., “The lottery ticket hypothesis for pre-trained BERT networks,” in Proc. Adv. Neural Inf. Process. Syst. (NIPS), vol. 33, 2020, pp. 15834–15846.

[52] M. Fey and J. E. Lenssen, “Fast graph representation learning with PyTorch geometric,” in Proc. Int. Conf. Learn. Represent. Workshop Learn. Graphs Manifolds, 2019, pp. 1–9.

[53] H. Zhou, A. Srivastava, H. Zeng, R. Kannan, and V. Prasanna, “Accelerating large scale real-time GNN inference using channel pruning,” Proc. VLDB Endowment, vol. 14, no. 9, pp. 1597–1605, May 2021.

[54] S. Narang et al., “Sparse training via boosting pruning plasticity with neuroregeneration,” in Proc. Adv. Neural Inf. Process. Syst., 2023, pp. 1–15.

[55] S. Narang, E. Ilsen, G. Diamos, and S. Sengupta, “Exploring sparsity in recurrent neural networks,” 2017, arXiv:1704.05119.

[56] M. Zhu and S. Gupta, “To prune, or not to prune: Exploring the efficacy of pruning for model compression,” 2017, arXiv:1710.01878.

[57] D. C. Mocanu, E. Mocanu, P. Stone, P. H. Nguyen, M. Gibescu, and A. Liotta, “Scalable training of artificial neural networks with adaptive sparse connectivity inspired by network science,” Nature Commun., vol. 9, no. 1, pp. 1–12, Jun. 2018.
Chuang Liu is currently pursuing the Ph.D. degree with the School of Computer Science, Wuhan University, Wuhan, China. His research interests include graph neural networks, graph pooling, and graph transformers.

Xueqi Ma is currently pursuing the Ph.D. degree with the School of Computing and Information Systems, The University of Melbourne, Melbourne, VIC, Australia. Her current research interests include pattern recognition and computer vision.

Yibing Zhan (Member, IEEE) received the bachelor’s and Ph.D. degrees from the School of Information Science and Technology, University of Science and Technology of China, Hefei, China, in 2012 and 2018, respectively. From 2018 to 2020, he served as an Associate Researcher with the School of Computer Science, Hangzhou Dianzi University, Hangzhou, China. He is currently with the JD Explore Academy, Beijing, China, as an Algorithm Scientist. He mainly explores multimodal learning and graph neural networks. He has authored many scientific articles in top conferences and journals, such as Conference on Neural Information Processing Systems (NeurIPS), the IEEE / CVF Computer Vision and Pattern Recognition Conference (CVPR), ACM International Conference on Multimedia (ACM MM), International Conference on Computer Vision (ICCV), and IEEE TRANSACTIONS ON MULTIMEDIA (TMM).

Liang Ding (Member, IEEE) received the Ph.D. degree from The University of Sydney, Sydney, NSW, Australia, in 2022. He works on deep learning for NLP, including language model pretraining, language understanding, generation, and translation. He has published more than 40 research papers in NLP/AI, including Annual Meeting of the Association for Computational Linguistics (ACL), Conference on Empirical Methods in Natural Language Processing (EMNLP), International Conference on Learning Representations (ICLR), and International Conference on Machine Learning (ICML). Dr. Ding was the Area (Session) Chair for ACL, Association for the Advancement of Artificial Intelligence (AAAI), and SIAM International Conference on Data Mining (SDM).

Dapeng Tao (Member, IEEE) received the B.E. degree from Northwestern Polytechnical University, Xi'an, China, in 1999, and the Ph.D. degree from the South China University of Technology, Guangzhou, China, in 2014. He is currently a Professor with the School of Information Science and Engineering, Yunnan University, Kunming, China. He has authored or coauthored more than 50 scientific articles. He has served for more than ten international journals, including IEEE TRANSACTIONS ON IMAGE PROCESSING (TIP), IEEE TRANSACTIONS ON NEURAL NETWORKS AND LEARNING SYSTEMS (TNNLS), IEEE TRANSACTIONS ON CYBERNETICS (TCYB), IEEE TRANSACTIONS ON MULTIMEDIA (TMM), and Information Sciences. His research interests include machine learning, computer vision, and robotics.

Bo Du (Senior Member, IEEE) received the Ph.D. degree in photogrammetry and remote sensing from the State Key Laboratory of Information Engineering in Surveying, Mapping and Remote Sensing, Wuhan University, Wuhan, China, in 2010. He is currently a Professor with the School of Computer Science and the Institute of Artificial Intelligence, Wuhan University, where he is also the Director of the National Engineering Research Center for Multimedia Software. He has more than 80 research articles published in the IEEE TRANSACTIONS ON IMAGE PROCESSING (TIP), IEEE TRANSACTIONS ON CYBERNETICS (TCYB), IEEE TRANSACTIONS ON PATTERN ANALYSIS AND MACHINE INTELLIGENCE (TPAMI), and IEEE TRANSACTIONS ON GEOSCIENCE AND REMOTE SENSING (TGRS). Fourteen of them are ESI hot articles or highly cited articles. His major research interests include machine learning, computer vision, and image processing. Dr. Du serves as an Associate Editor for Neural Networks, Pattern Recognition, and Neurocomputing. He also serves as a Reviewer for 20 Science Citation Index (SCI) magazines, including IEEE TPAMI, IEEE TCYB, IEEE TGRS, IEEE TIP, and IEEE GEOSCIENCE AND REMOTE SENSING LETTERS (GRSL).

Wenbin Hu is currently a Professor with the School of Computer Science, Wuhan University, Wuhan, China. He has authored or coauthored more than 60 refereed journal articles and conference papers, including the IEEE TRANSACTIONS ON KNOWLEDGE AND DATA ENGINEERING, the IEEE TRANSACTIONS ON MOBILE COMPUTING, IEEE TRANSACTIONS ON INTELLIGENT TRANSPORTATION SYSTEMS, ACM Transactions on Knowledge Discovery from Data, Special Interest Group on Knowledge Discovery in Data (SIGKDD), the IEEE International Conference on Data Mining (ICDM), Association for the Advancement of Artificial Intelligence (AAAI), and International Joint Conference on Artificial Intelligence (IJCAI). His research interests include data mining and machine learning, uncertainty artificial intelligence, and complex networks.

Danilo P. Mandic (Fellow, IEEE) received the Ph.D. degree in nonlinear adaptive signal processing from Imperial College London, London, U.K., in 1999. He is currently a Professor in signal processing with Imperial College London. His publication record includes two research monographs titled Recurrent Neural Networks for Prediction: Learning Algorithms, Architectures and Stability (Wiley, First Edition, August 2001) and Complex Valued Nonlinear Adaptive Filters: Noncircularity, Widely Linear and Neural Models (Wiley, First Edition, April 2009), an edited book titled Signal Processing Techniques for Knowledge Extraction and Information Fusion (Springer, 2008), and a two-volume research monograph titled Tensor Networks for Dimensionality Reduction and Large Scale Optimization (Now Publishers, 2016 and 2017). Dr. Mandic received the Denis Gabor Award for Outstanding Achievements in Neural Engineering by the International Neural Networks Society, the 2018 Best Paper Award in the IEEE Signal Processing Magazine for his article on tensor decompositions, and the President Award for Excellence in Postgraduate Supervision at Imperial College London. He has been an Associate Editor of the IEEE TRANSACTIONS ON SIGNAL PROCESSING, the IEEE TRANSACTIONS ON SIGNAL AND INFORMATION PROCESSING OVER NETWORKS.