# Graph Sparsification via Meta-Learning

**Guihong Wan, Harsha Kokel**
The University of Texas at Dallas
800 W. Campbell Road, Richardson, Texas 75080
{Guihong.Wan, hkokel}@utdallas.edu

**Abstract**

We present a novel graph sparsification approach for semi-supervised learning on undirected attributed graphs. The main challenge is to retain few edges while minimize the loss of node classification accuracy. The task can be mathematically formulated as a bi-level optimization problem. We propose to use meta-gradients, which have traditionally been used in meta-learning, to solve the optimization problem, essentially treating the graph adjacency matrix as hyperparameter to optimize. Experimental results show the effectiveness of the proposed approach. Remarkably, with the resulting sparse and light graph, in many cases the classification accuracy is significantly improved.

## 1 Introduction

Neural Networks designed for graph-structured data have recently shown tremendous success in various domains, ranging from social analysis, bioinformatics, natural language processing to computer vision. See e.g., Zhang et al. (2018) and Wu et al. (2020) for recent surveys. With the graph structure and node attributes as inputs, Graph Neural Network (GNN) can focus on different learning tasks, including node level, edge level and graph level tasks, and can be trained in a supervised, semi-supervised, or unsupervised way. We investigate the semi-supervised node classification task on undirected graphs. In such task, given an attributed graph with a small subset of labeled nodes, the goal is to predict the labels of unlabeled nodes. See, e.g., Kipf and Welling (2017); Veličković et al. (2018); Li et al. (2020).

Graph sparsification is a fundamental problem in graph analysis. The goal is to approximate a given graph by a sparse graph so that important properties (i.e., graph spectrum) are approximately preserved. See, e.g., Spielman and Srivastava (2011); Spielman and Teng (2011); Fung et al. (2019). In this work, we focus on the sparsification of a graph that preserves the node classification accuracy of the unlabeled nodes. Such sparsification fosters more effective understanding of information propagation on graphs and hence can be beneficial in wide range of graph-based deep learning applications.

Gradient-based meta-learning is a well known approach for learning-to-learn in numerous domains. It has several possible formulations. One way is to frame it as a bi-level optimization procedure in which the "inner" optimization represents adaptation to a specific task, and the "outer" objective is the meta-learning objective. See, e.g., Finn, Abbeel, and Levine (2017); Zügner and Günnemann (2019); Rajeswaran et al. (2019). In a well-known paper, Model-Agnostic Meta-Learning (MAML) (Finn, Abbeel, and Levine 2017), such a formulation is used to learn the initial parameters of stochastic gradient descent models. One of the key challenges in gradient-based meta-learning is that the computation of gradients of the meta objectives needs to backpropagate through the inner optimization. If the inner optimization goes on for many steps, the gradients of the meta objectives vanish, and it also imposes computational and memory burdens. MAML proposes a first-order approximation of the proposed approach. In Rajeswaran et al. (2019), iMAML is proposed to further solve this problem, which depends only on the solution to the inner optimization. Zügner and Günnemann (2019) shows how to use meta-learning to solve the bi-level optimization for training-time adversarial attacks on graphs.

### Our approach

Inspired by these gradient-based meta-learning literature, we propose to use meta-learning to reduce the number of edges in the graph, concentrating on node classification task in semi-supervised setting. Essentially, by treating the graph structure as meta-parameter, the underlying optimization problem is solved via meta-gradients. In our case, the inner optimization is to train the model over labeled nodes for predicting labels of unlabeled nodes, and the outer meta-objective aims to eliminate edges with minimal impact on the classification accuracy. The idea of using meta-learning in this context appears to be novel, and can most likely be applied in related situations, like feature selection or graph classification task, that are not explicitly discussed in this paper.

## 2 Related Work

A variety of semi-supervised learning methods exist in the literature, like self-training and graph-based methods. See e.g., Zhu (2005); Sheikhpour et al. (2017) for surveys. The main idea of self-training is to first train a classifier on labeled data and then apply it to classify the unlabeled data.

A subset of the unlabeled data, together with their predicted labels are used in later learning process. See, e.g., Zhu and Goldberg (2009). Graph-based methods aim to construct a graph connecting similar observations; label information propagates through the graph from labeled to unlabeled nodes. See, e.g., Kipf and Welling (2017); Veličković et al. (2018). A variant of the proposed approach uses the self-training method to compute the sparsifier's loss on accuracy, discussed in Section 3.

**Graph Sparsification.** One of the popular approaches is the spectral sparsification technique. Spielman and Teng (2011) introduced a spectral sparsification method based on the similarity of graph Laplacians. In the paper (Spielman and Srivastava 2011), the authors proposed to sample edges in proportion to the effective resistance of an edge. With this sampling approach, the graph Laplacian spectrum and resistance distances between vertices are approximately preserved with high probability. Imre et al. (2020) presented a spectrum-preserving sparsification algorithm for visualizing big graph data. Other approaches, like cut sparsification and structure-preserving sparsification, are also proposed. Cut sparsification methods, e.g., (Fung et al. 2019), ensure that the total weight of cuts in the resulting graph approximates that of cuts in the original graph. For structure-preserving sparsification methods, see e.g., (Satuluri, Parthasarathy, and Ruan 2011; Hamann et al. 2016). While properties preserved in these works are significantly important, they might not be necessary for the graph classification or node classification tasks. Rather, it is important to preserve the classification accuracy. We hypothesize that directly targeting the accuracy measure would work better for node classification, and formulate a meta-learning based graph sparsification algorithm.

## 3 Problem Formulation and Notations

The problem addressed in this work is as follows. Given an undirected attributed graph with labeled and unlabeled nodes, the goal is to achieve the target graph sparsity while minimize the node classification error over unlabeled nodes.

Let $G{=}(A, X)$ be an undirected attributed graph with adjacency matrix $A \in \{0,1\}^{N\times N}$ and node attribute matrix $X \in \mathbb{R}^{N\times D}$, where $N$ is the number of nodes and $D$ is the dimension of node features. Let $V$ be the set of all nodes of graph $G$ which is the union of the subset of labeled nodes $V_L \subseteq V$ and the subset of unlabeled nodes $V_U{=}V \setminus V_L$. Each node in $V_L$ is assigned one class in $C = \{c_1, \dots, c_K\}$. Let $Y_L$ be the labels of labeled nodes in $V_L$. Let $Y_U$ be the labels of unlabeled nodes in $V_U$, which is unknown.

Given graph $G$ and labels $Y_L$, the goal of semi-supervised learning for node classification is to learn a function $f_\theta$, which maps each node $v \in V$ to exactly one of the $K$ classes in $C$. The parameters $\theta$ of the function $f_\theta$ are learned by minimizing a loss function $\mathcal{L}_{\text{train}}$ on the labeled nodes:

$$\theta^* = \arg\min_\theta \mathcal{L}_{\text{train}}(f_\theta(G), Y_L). \tag{1}$$

Let $M$ be the number of nonzero values in the adjacency matrix $A$. Then the number of edges of the undirected graph $G$ is $\frac{M}{2}$. Let $\hat{M} < M$ be the target number of nonzero values in the adjacency matrix. The graph sparsification problem can be mathematically formulated as a bi-level optimization problem:

$$\begin{aligned} \hat{G}^* &= \min_{\hat{G}\in\Phi(G)} \mathcal{L}_{\text{sps}}(f_{\theta^*}(\hat{G}), Y_U) \\ &s.t. \quad \theta^* = \arg\min_\theta \mathcal{L}_{\text{train}}(f_\theta(\hat{G}), Y_L). \end{aligned} \tag{2}$$

The $\Phi(G)$ is the admissible space of $\hat{G}$. The achieved graph sparsity is $\frac{\hat{M}}{N(N-1)}$. The $\mathcal{L}_{\text{sps}}$ is the loss function that the sparsifier aims to optimize. In the bi-level meta-learning setup, the constraint in Equation 2, which is same as Equation 1, is the "inner" loop; the parameters $\theta$ are the task-specific parameters. The minimizer for $\mathcal{L}_{\text{sps}}$ is the "outer" loop; the parameters $\hat{G} \in \Phi(G)$ are the meta-parameters.

### Options for $\mathcal{L}_{\text{sps}}$

In the task, the sparsifier tries to decrease the classification error over unlabeled nodes. However, the labels for nodes in $V_U$ are not available. The sparsifier cannot directly optimize the $\mathcal{L}_{\text{sps}}$. There are several options to approximate $\mathcal{L}_{\text{sps}}$.

The first option is $\mathcal{L}_{\text{sps}} \approx \mathcal{L}_{\text{train}}$. If a model has a high training error, it typically cannot generalize well for the test set; however the opposite is not true due to overfiting. In our case, the sparsifier tries to simplify the graph. If there is overfiting, the classifiers may work better on the simplified graph. This is confirmed experimentally. As discussed in Section 2, self-training is a well-known method to augment the labeled data in semi-supervised learning. In our case, the sparsifier can train a classifier on labeled data to estimate the labels of unlabeled nodes. We refer the predicted labels as $\hat{Y}_U$. Using the predicted labels the sparsifier can perform self-learning and compute $\mathcal{L}_{\text{sps}}$ on the unlabeled nodes. This gives us the second option $\mathcal{L}_{\text{sps}} \approx \mathcal{L}_{\text{self}}$, where $\mathcal{L}_{\text{self}} = \mathcal{L}(f_{\theta^*}(\hat{G}), \hat{Y}_U)$. Another option is $\mathcal{L}_{\text{sps}} \approx \mathcal{L}_{\text{both}}$, which is computed from $Y_L$ and $\hat{Y}_U$. In the Section 6, we experimentally compare all the three options outlined above.

## 4 Meta-Gradients

We use meta-gradients to solve the bi-level optimization problem in Equation 2. Gradient-based meta-learning has traditionally been used to optimize hyperparameters, e.g., learning the initial parameters of a model (Finn, Abbeel, and Levine 2017; Rajeswaran et al. 2019) and finding hyperparameters (Bengio 2000; Zügner and Günnemann 2019).

In our case, we treat the graph structure matrix as hyperparameter. We are interested in finding the graph structure matrix satisfying the target sparsity. We compute the gradient of the sparsifier's loss w.r.t the hyperparameter:

$$\begin{aligned} \nabla_{\hat{G}}^{\text{meta}} &:= \nabla_{\hat{G}} \mathcal{L}_{\text{sps}}(f_{\theta^*}(\hat{G}), Y_U), \\ &s.t. \quad \theta^* = \arg\min_\theta \mathcal{L}_{\text{train}}(f_\theta(\hat{G}), Y_L). \end{aligned} \tag{3}$$

The meta-gradient indicates how the sparsifier loss $\mathcal{L}_{\text{sps}}$ (the loss of classification accuracy on unlabeled nodes) will change after training on the simplified graph.

The task in Equation 1 corresponds to multiple steps of gradient descent starting from some initial parameters $\theta_0$ with the learning rate $\alpha$:

$$\theta_{t+1} = \theta_t - \alpha \nabla_{\theta_t} \mathcal{L}_{\text{train}}(f_{\theta_t}(\hat{G}), Y_L). \tag{4}$$

We approximate $\theta^*$ by $\theta_T$ obtained after applying gradient descent for $T$ steps. Without loss of generality, the sparsifier's loss is written as $\mathcal{L}_{\text{sps}}(f_{\theta_T}(\hat{G}))$. The meta-gradient can be calculated as:

$$\begin{aligned}\nabla_{\hat{G}}^{\text{meta}} &= \nabla_{\hat{G}} \mathcal{L}_{\text{sps}}(f_{\theta_T}(\hat{G})) \\ &= \frac{\partial \mathcal{L}_{\text{sps}}(f_{\theta_T}(\hat{G}))}{\partial f} \Big[ \frac{\partial f_{\theta_T}(\hat{G})}{\partial \hat{G}} + \frac{\partial f_{\theta_T}(\hat{G})}{\partial \theta_T} \frac{\partial \theta_T}{\partial \hat{G}} \Big] \\ &= \nabla_f \mathcal{L}_{\text{sps}}(f_{\theta_T}(\hat{G})) [\nabla_{\hat{G}} f_{\theta_T}(\hat{G}) + \nabla_{\theta_T} f_{\theta_T}(\hat{G}) \nabla_{\hat{G}} \theta_T].\end{aligned} \tag{5}$$

The parameters $\theta_T$ depend on the graph $\hat{G}$. Thus, the derivative w.r.t the graph has to chain back until the initial parameters $\theta_0$. In order to calculate the gradient $\nabla_{\hat{G}} \theta_T$, we need to calculate $\nabla_{\hat{G}} \theta_t$ for $t = T-1, \ldots, 1$ as follows:

$$\begin{aligned}&\nabla_{\hat{G}} \theta_{t+1} = \nabla_{\hat{G}} \theta_t - \alpha \nabla_{\hat{G}} \nabla_{\theta_t} \mathcal{L}_{\text{train}}(f_{\theta_t}(\hat{G})) \\ &\nabla_{\hat{G}} \theta_t = \nabla_{\hat{G}} \theta_{t-1} - \alpha \nabla_{\hat{G}} \nabla_{\theta_{t-1}} \mathcal{L}_{\text{train}}(f_{\theta_{t-1}}(\hat{G})) \\ &\cdots \\ &\nabla_{\hat{G}} \theta_1 = \nabla_{\hat{G}} \theta_0 - \alpha \nabla_{\hat{G}} \nabla_{\theta_0} \mathcal{L}_{\text{train}}(f_{\theta_0}(\hat{G})).\end{aligned} \tag{6}$$

Observe that it is expensive to calculate the gradient of the meta objective using the above meta-gradients due to the need of differentiating through the inner loop learning process. However, approaches to approximate the meta-gradients are proposed, e.g., (Finn, Abbeel, and Levine 2017; Rajeswaran et al. 2019).

Now we can conduct the following meta-gradient descent:

$$\hat{G}^{k+1} = \hat{G}^k - \beta \nabla_{\hat{G}^k} \mathcal{L}_{\text{sps}}(f_{\theta^*}(\hat{G}^k)), \tag{7}$$

where $\hat{G}^0 = G$. The $\hat{G}^*$ is obtained when the target graph sparsity is achieved. However, the update in Equation 7 is not possible with discrete graph-structured data. We discuss this problem for graph sparsification in Section 5.

## 5 Graph Sparsification via Meta-Gradients

An undirected attributed graph $G$=$(A, X)$ contains two components: the adjacency matrix $A \in \{0,1\}^{N \times N}$ and the node attribute matrix $X \in \mathbb{R}^{N \times D}$. In order to reduce the graph in terms of edges, we compute the meta-gradients w.r.t the adjacency matrix $A$ with unchanged the attribute matrix $X$. Analogously to Equation (3), the meta-gradients are defined as follows:

$$\begin{aligned}\nabla_{\hat{A}}^{\text{meta}} :=& \nabla_{\hat{A}} \mathcal{L}_{\text{sps}}(f_{\theta^*}(\hat{A}, X), Y_U) \\ & s.t. \quad \theta^* = \arg\min_{\theta} \mathcal{L}_{\text{train}}(f_\theta(\hat{A}, X), Y_L).\end{aligned} \tag{8}$$

The adjacency matrix is modified in meta-gradient update in Equation (7).

**Algorithm 1 Graph sparsification via meta-gradients**

**Input:** Graph $G = (A, X)$; labels $Y_L$; number of edges to delete $\zeta$; number of training steps $T$; learning rate $\alpha$.
**Output:** $\hat{G}^* = (\hat{A}^*, X)$

```
1:  $\hat{Y}_U$ ← estimated labels of unlabeled nodes using self-training;
2:  $\hat{A} \leftarrow A$;
3:  while $\zeta > 0$ do
4:      $\theta_0$ ← initialize randomly;
5:      for $t$ in $0 \ldots T-1$ do
6:          $\theta_{t+1} = \theta_t - \alpha \nabla_{\theta_t} \mathcal{L}_{\text{train}}(f_{\theta_t}(\hat{A}, X), Y_L)$;
7:      end for
8:      $\nabla_{\hat{A}}^{\text{meta}} \leftarrow \nabla_{\hat{A}} \mathcal{L}_{\text{self}}(f_{\theta_T}(\hat{A}, X), \hat{Y}_U)$;
9:      $S = \nabla_{\hat{A}}^{\text{meta}} \odot \hat{A}$;
10:     $e^*$ ← the maximum entry $(i,j)$ in $S(i,j)$ that satisfies the constraints $\Phi(G)$;
11:     $\hat{A}$ ← remove edge $e^*$;
12:     $\zeta$ −= 1;
13: end while
14: $\hat{G}^* \leftarrow (\hat{A}, X)$;
15: return $\hat{G}^*$.
```

A two-layer graph convolutional network (GCN) (Kipf and Welling 2017) is used to perform the reduction and evaluate the performance of the reduced graph for node classification:

$$f_\theta(A, X) = \text{softmax}(A' \delta(A' X W_1) W_2),$$

where $\delta$ is the activation function, $A' = D^{-1/2} \tilde{A} D^{-1/2}, \tilde{A} = A + I$, $D$ is the degree matrix, $\theta$ is the set of learnable parameters in $W_1$ and $W_2$. However our proposed approach is model-agnostic. Other graph networks, like graph attention networks (GAT) (Veličković et al. 2018), can be used.

However, we are interested in the 0/1 problem, that is, an edge is either kept or deleted. We cannot simply perform the gradient decent as follows:

$$\hat{A}^{k+1} = \hat{A}^k - \beta \nabla_{\hat{A}^k}^{\text{meta}}, \text{ with } \hat{A}^0 = A. \tag{9}$$

Instead we introduce a score matrix $S = \nabla_{\hat{A}}^{\text{meta}} \odot \hat{A}$, performing element-wise production between the meta-gradients and the adjacency matrix. We greedily pick the edge between the node $i$ and $j$: $e(i,j)$ with the highest score to be eliminated:

$$e^* = \underset{\substack{e(i,j) \in \hat{A} \\ e(i,j) \in \Phi(G)}}{\arg\max} \; S(i,j). \tag{10}$$

The edge $e(i,j)$ should fulfill the constraint $\Phi(G)$ that the deletion of $e(i,j)$ should not lead to singleton nodes. In Algorithm 1, we summarize the steps to sparsify graphs via meta-gradients and self-training.

**Interpretation of the score matrix.** For graph sparsification, we only consider meta-gradients with edges ($\hat{A}(i,j) = 1$). The values of meta-gradients where no edges are zeroed out by the element-wise production between the meta-gradients and the adjacency matrix. Second, we prefer the

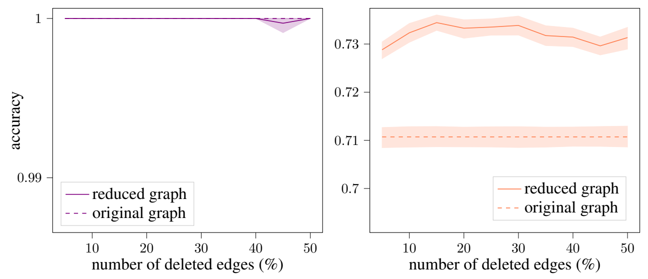

Figure 1: Accuracy when performing with $\mathcal{L}_{\text{train}}$ on CITESEER. Left: train-set; Right: test-set.

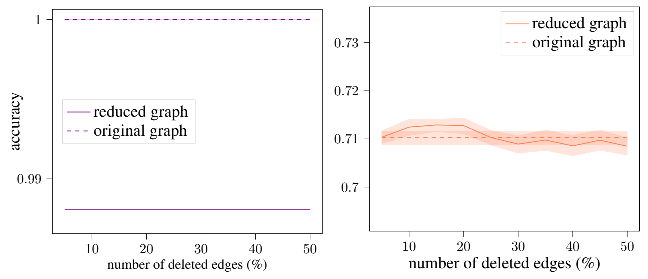

Figure 2: Accuracy when performing with $\mathcal{L}_{\text{self}}$ on CITESEER. Left: train-set; Right: test-set.

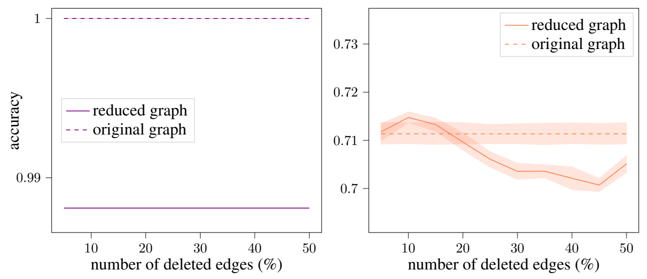

Figure 3: Accuracy when performing with $\mathcal{L}_{\text{both}}$ on CITESEER. Left: train-set; Right: test-set.

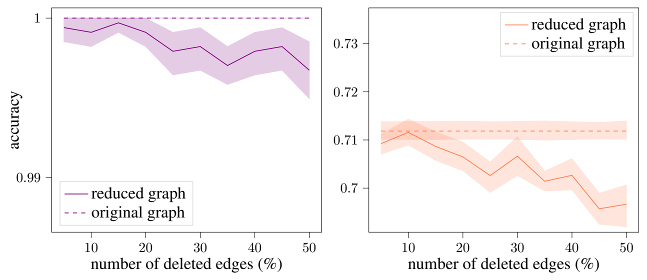

Figure 4: Accuracy when using random selection on CITESEER. Left: train-set; Right: test-set.

positive meta-gradients, as we want to modify $\hat{A}(i,j)$ from 1 to 0. In addition, the meta-gradient indicates how the loss will change after training on the modified graph. The above statements explain the idea behind the score matrix.

## 6 Experimental Results

We evaluate the performance of the proposed approach on two well-known datasets: CITESEER (Sen et al. 2008) and CORA-ML (McCallum et al. 2000). The statistics of datasets are shown in Table 1.

A two-layer graph convolutional network (GCN) (Kipf and Welling 2017) is used to perform the sparsification and evaluate the performance of the reduced graph for node classification. Followed from Zügner and Günnemann (2019), the hidden size of GCN is 16. To compute the accuracy, the classifiers are trained for 100 iterations and all experiments are repeated for 20 times. The graphs are split into labeled (10%) and unlabeled (90%) nodes. The labels of the unlabeled nodes are only used for node classification accuracy during testing phase. We compute the meta-gradients by using gradient descent with momentum for $T = 50$ iterations.

In Section 3, three methods are proposed to approximate $\mathcal{L}_{\text{sps}}$. The results on CITESEER dataset are shown in Figures 1, 2 and 3. The results on CORA-ML dataset are shown in Figures 5, 6 and 7. In addition, the results using random selection are shown in Figures 4 and 8.

| **Dataset** | $N_{\text{LCC}}$ | $E_{\text{LCC}}$ | D | K |
|---|---|---|---|---|
| CORA-ML | 2,810 | 7,981 | 2,879 | 7 |
| CITESEER | 2,110 | 3,668 | 3,703 | 6 |

Table 1: We consider the largest connected component (LCC). $N_{\text{LCC}}$: number of nodes; $E_{\text{LCC}}$: number of edges.

Surprisingly, using $\mathcal{L}_{\text{train}}$ in Figure 1 improves the test accuracy, even 50% edges are deleted. From the training accuracy, the classifier might overfit when performing with the original graph.

In Figure 6, using $\mathcal{L}_{\text{self}}$, the edge deletion first improves the test accuracy, and then the accuracy decreases as more edges are deleted. The training accuracy shows that the initial edge deletion helps the classifier to fit the data. The result shows that about 20% edges can be eliminated without hurting the classification accuracy.

Our experiments show that when the classifier overfits the original graph, using $\mathcal{L}_{\text{train}}$ performs better. When the classifier underfits the original graph, using $\mathcal{L}_{\text{self}}$ helps the classifier fit the data.

## 7 Discussion and Future Work

The experimental results of using meta-gradients to reduce the edges for graphs are very encouraging and interesting. We need to further compare our approach with the conventional graph sparsification techniques. We need to investigate deeply by removing more number edges and experimenting on different datasets. We also need to evaluate the transferability of our technique by using various gradient descent models. The proposed approach can most likely be applied in related situations, like feature selection or graph classification, and can be generalized for attributed edges, that are not explicitly discussed in this paper.

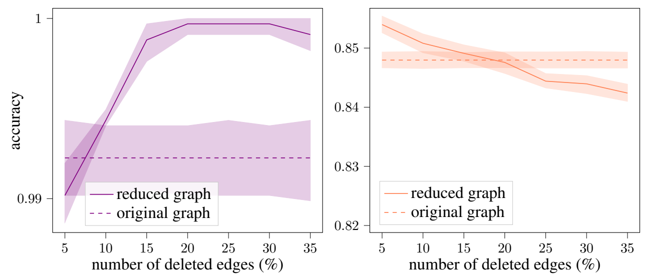

Figure 5: Accuracy when performing with $\mathcal{L}_{\text{train}}$ on CORA-ML. Left: train-set; Right: test-set.

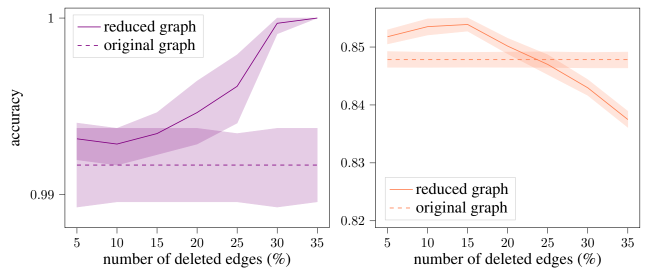

Figure 6: Accuracy when performing with $\mathcal{L}_{\text{self}}$ on CORA-ML. Left: train-set; Right: test-set.

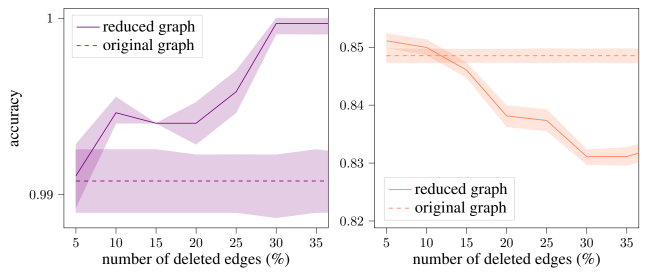

Figure 7: Accuracy when performing with $\mathcal{L}_{\text{both}}$ on CORA-ML. Left: train-set; Right: test-set.

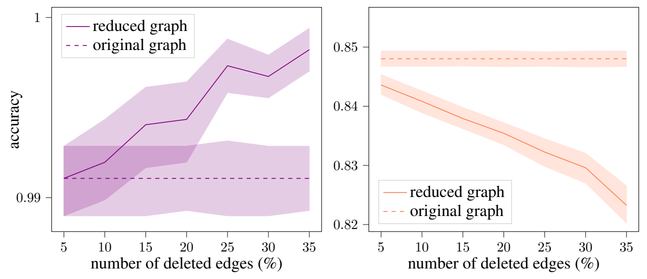

Figure 8: Accuracy when using random deletion on CORA-ML. Left: train-set; Right: test-set.

## References

Bengio, Y. 2000. Gradient-based optimization of hyperparameters. *Neural computation* 12(8): 1889–1900.

Finn, C.; Abbeel, P.; and Levine, S. 2017. Model-agnostic meta-learning for fast adaptation of deep networks. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, 1126–1135. JMLR. org.

Fung, W.-S.; Hariharan, R.; Harvey, N. J.; and Panigrahi, D. 2019. A general framework for graph sparsification. *SIAM Journal on Computing* 48(4): 1196–1223.

Hamann, M.; Lindner, G.; Meyerhenke, H.; Staudt, C. L.; and Wagner, D. 2016. Structure-preserving sparsification methods for social networks. *Social Network Analysis and Mining* 6: 22.

Imre, M.; Tao, J.; Wang, Y.; Zhao, Z.; Feng, Z.; and Wang, C. 2020. Spectrum-preserving sparsification for visualization of big graphs. *Computers & Graphics* 87: 89–102.

Kipf, T. N.; and Welling, M. 2017. Semi-Supervised Classification with Graph Convolutional Networks. In *International Conference on Learning Representations, ICLR 2017*.

Li, K.; Feng, Y.; Gao, Y.; and Qiu, J. 2020. Hierarchical graph attention networks for semi-supervised node classification. *Appl. Intell.* 50(10): 3441–3451.

McCallum, A. K.; Nigam, K.; Rennie, J.; and Seymore, K. 2000. Automating the construction of internet portals with machine learning. *Information Retrieval* 3(2): 127–163.

Rajeswaran, A.; Finn, C.; Kakade, S. M.; and Levine, S. 2019. Meta-learning with implicit gradients. In *Advances in Neural Information Processing Systems*, 113–124.

Satuluri, V.; Parthasarathy, S.; and Ruan, Y. 2011. Local graph sparsification for scalable clustering. In *Proceedings of the 2011 ACM SIGMOD International Conference on Management of data*, 721–732.

Sen, P.; Namata, G.; Bilgic, M.; Getoor, L.; Galligher, B.; and Eliassi-Rad, T. 2008. Collective classification in network data. *AI magazine* 29(3): 93–93.

Sheikhpour, R.; Sarram, M. A.; Gharaghani, S.; and Chahooki, M. A. Z. 2017. A survey on semi-supervised feature selection methods. *Pattern Recognition* 64: 141–158.

Spielman, D. A.; and Srivastava, N. 2011. Graph sparsification by effective resistances. *SIAM Journal on Computing* .

Spielman, D. A.; and Teng, S.-H. 2011. Spectral sparsification of graphs. *SIAM Journal on Computing* 40(4): 981–1025.

Veličković, P.; Cucurull, G.; Casanova, A.; Romero, A.; Liò, P.; and Bengio, Y. 2018. Graph Attention Networks. In *International Conference on Learning Representations*.

Wu, Z.; Pan, S.; Chen, F.; Long, G.; Zhang, C.; and Philip, S. Y. 2020. A comprehensive survey on graph neural networks. *IEEE Transactions on Neural Networks and Learning Systems* .

Zhang, S.; Tong, H.; Xu, J.; and Maciejewski, R. 2018. Graph convolutional networks: Algorithms, applications and open challenges. In *International Conference on Computational Social Networks*, 79–91. Springer.

Zhu, X.; and Goldberg, A. B. 2009. Introduction to semi-supervised learning. *Synthesis lectures on artificial intelligence and machine learning* 3(1): 1–130.

Zhu, X. J. 2005. Semi-supervised learning literature survey. Technical report, University of Wisconsin-Madison Department of Computer Sciences.

Zügner, D.; and Günnemann, S. 2019. Adversarial Attacks on Graph Neural Networks via Meta Learning. In *7th International Conference on Learning Representations, ICLR 2019*.