Edge Sparsification for Graphs via Meta-Learning

Guihong Wan
Supervised by Prof. Haim Schweitzer
Department of Computer Science
The University of Texas at Dallas
Richardson, Texas 75080
Guihong.Wan@utdallas.edu

***Abstract*—We present a novel edge sparsification approach for semi-supervised learning on undirected and attributed graphs. The main challenge is to retain few edges while minimizing 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, specifically treating the graph adjacency matrix as hyperparameters 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.**

***Index Terms*—Graph Sparsification, Meta-Learning, Meta-Gradients, Node Classification, Graph Neural Network**

I. Introduction

Neural Networks designed for graph structured data have recently shown tremendous success in various domains, ranging from social analysis, computer vision to natural language processing. See e.g., [1]–[4]. With the graph structure and node attributes as inputs, Graph Neural Network (GNN) (e.g., [5], [6]) can work 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 learning on undirected and attributed graphs where labels are only available for a small subset of nodes. Given an attributed graph with a subset of labeled nodes, the goal is to predict the labels of unlabeled nodes. See, e.g., [7]–[9].

Graph sparsification is a fundamental problem in graph analysis. For example, the simplified structure fosters effective understanding of the information propagation on graphs. The goal is to approximate a given graph by a sparse graph on the same set of nodes so that important properties (i.e., graph spectrum) are approximately preserved. See, e.g., [10]–[12]. In this work, we focus on the edge sparsification of a given graph in which the loss of node classification accuracy of the unlabeled nodes is minimized.

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., [13]–[17]. In the well-known MAML paper [13], 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. The authors introduce a first-order approximation of the proposed approach. In [16], iMAML is introduced to further solve this problem, which depends only on the solution to the inner optimization. Zügner and Günnemann [15] use meta-learning to solve the bi-level optimization problem for training-time adversarial attacks on graphs.

Our Approach

Inspired by these gradient-based meta-learning works, we propose to use meta-learning to eliminate the edges of graphs, concentrating on the node classification task in semi-supervised setting. Specifically, by treating the graph adjacency matrix as meta-parameters, the underlying bi-level optimization problem is solved via meta-gradients. In our case, the inner optimization is to train the model over labeled nodes for predicating labels of unlabeled nodes, and the outer meta-objective is for the sparsifier which aims to delete edges but preserve the node 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.

II. Related Works

A variety of semi-supervised learning methods exist in the literature. See e.g., [18], [19] for surveys. One of the important methods is self-training. The main idea is to first train a classifier on labeled data and then apply it to predict the labels of unlabeled data. A subset of the unlabeled data, together with their predicted labels are used in later learning process. See, e.g., [20], [21]. A variant of the proposed approach uses the self-training method to compute the sparsifier's loss on the node classification accuracy, discussed in Section III.

Graph Sparsification. Previously researchers have developed approaches for sparsifying graphs, but not with the same goals as ours. One popular approach is spectral sparsification technique (see e.g., [10], [11]), which preserves graph Laplacian spectrum. Spielman and Teng [11] introduced a spectral

sparsification method based on the similarity of graph Laplacians. In the paper [10], the authors proposed to sample edges in proportion to the effective resistance of an edge, preserving the graph Laplacian spectrum with high probability. Another approach is structure-preserving sparsification methods. See e.g., [22]–[26]. For example, Local Degree [22] preserves edges leading to local hub nodes. Forest Fire node sampling was originally proposed in [23]. The authors of [22] modified it for edge sampling. Other approaches are also proposed, like cut sparsification (e.g., [12]), which ensures that the total weight of cuts in the resulting graph approximates that of cuts in the original graph.

While properties preserved in these works are important, it might not be necessary for the graph classification or node classification problems. In our case, it is rather important to maintain the node classification accuracy while reducing the graph to a sparser graph. It might be more effective by directly targeting on the classification accuracy measure for sparsification. With this hypothesis, we propose a meta-learning based graph sparsification algorithm.

III. Problem Formulation and Notations

The problem addressed in this work is as follows. Given an undirected attributed graph with labeled nodes and unlabeled nodes, and the target graph sparsity, the goal is to achieve the target graph sparsity while minimizing 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 $C = \{c_1, \ldots, c_K\}$. 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_θ, which maps each node $v \in V$ to exactly one of the K classes in C. The parameters θ of the function f_θ 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:

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

Here $\Phi(G)$ is the admissible space of $\hat{G}$. The achieved graph sparsity is $1 - \frac{\hat{M}}{N(N-1)}$. The $\mathcal{L}_{\text{sps}}$ is the loss function that the sparsifier aims to optimize. In the bi-level meta-learning setup, the second part of Equation (2), which is same as Equation (1), is the "inner" loop; the parameters θ are the task-specific parameters. The minimizer for $\mathcal{L}_{\text{sps}}$ in first part of Equation (2) is the "outer" loop; the parameters $\hat{G} \in \Phi(G)$ are the meta-parameters to be found. Similar formulation is used in [15] for training-time adversarial attacks on graphs.

Options for $\mathcal{L}_{\text{sps}}$: The sparsifier tries to decrease the node classification error over unlabeled nodes. However, the labels for nodes in V_U are not available. The sparsifier cannot directly optimize $\mathcal{L}_{\text{sps}}$. There are two options to approximate $\mathcal{L}_{\text{sps}}$. The first option is $\mathcal{L}_{\text{sps}} \approx \mathcal{L}_{\text{train}}$. If a classifier 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 classifier may work better on the simplified graph. This is confirmed experimentally. As discussed in Section II, 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 $\mathcal{L}_{\text{self}} = \mathcal{L}(f_{\theta^*}(\hat{G}), \hat{Y}_U)$. In the Section VII, we experimentally compare the above options.

IV. 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 [13], [16] and finding hyperparameters [15], [27].

In our case, we treat the graph structure matrix as hyper-parameters. 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 hyperparameters:

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

The meta-gradients indicate 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 θ_0 with the learning rate α:

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

We approximate θ^* by θ_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-gradients can be calculated as follows:

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

The parameters θ_T depend on the graph $\hat{G}$. Thus, the derivative w.r.t the graph has to chain back until the initial parameters θ_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:

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

Observe that it is expensive to calculate the gradients 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., [13], [16].

Now we can conduct the following meta-gradient descent:

$$\hat{G}^{k+1} = \hat{G}^k - \beta\nabla_{\hat{G}^k}\mathcal{L}_{\text{sps}}(f_{\theta^*}(\hat{G}^k)), \text{ with } \hat{G}^0 = G, \tag{7}$$

where β is the learning rate. 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 V.

V. 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. The meta-gradients in Equation (3) can be written as follows:

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

Algorithm 1 Graph sparsification via meta-gradients

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

```
Ŷ_U ← labels of unlabeled nodes using self-training;
Â ← A;
while ζ > 0 do
    θ_0 ← random initialization;
    for t in 0 ... T − 1 do
        θ_{t+1} = θ_t − α∇_{θ_t} L_train(f_{θ_t}(Â, X), Y_L);
    end for
    ∇^meta_Â ← ∇_Â L_self(f_{θ_T}(Â, X), Ŷ_U);
    S = ∇^meta_Â ⊙ Â;
    e* ← the maximum entry (i, j) in S that satisfies the
    constraints Φ(G);
    Â ← remove edge e*;
    ζ −= 1;
end while
Ĝ* ← (Â, X);
return Ĝ*.
```

A two-layer graph convolutional network (GCN) [7] is used to perform the reduction and evaluate the performance of the reduced graph for node classification:

$$f_\theta(A, X) = \text{softmax}(A'\delta(A'XW_1)W_2),$$

where δ is the activation function, $A' = D^{-1/2}\tilde{A}D^{-1/2}, \tilde{A} = A + I$, I is the identity matrix, D is the degree matrix, θ is the set of learnable parameters in W_1 and W_2. However our proposed approach is model-agnostic. Other graph networks, like GraphSAGE [28], 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:

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

Instead we introduce a score matrix $S = \nabla^{\text{meta}}_{\hat{A}} \odot \hat{A}$, 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:

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

The edge $e(i, j)$ should fulfill the constraints $\Phi(G)$. For example, 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. First, 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 positive meta-gradients, as we want to modify $\hat{A}(i, j)$ from 1 to 0. In addition, the meta-gradients indicate how the loss will change after training on the modified graph. For weighted graphs, we can learn an indicator matrix initialized as 1 if there is an edge. The above statements explain the idea behind the score matrix.

VI. Complexity Analysis

For exact meta-gradients, we need to store the weights θ for T gradient descent steps, costing memory $O(T\cdot|\theta|)$. For graph sparsification, we only consider meta-gradients with edges. Let M be the number of edges. The memory complexity is $O(M + T\cdot|\theta|)$. The second-order derivatives at each step in the meta-gradient formulation can be computed in $O(M)$. The overall time complexity is $O(T \cdot M\zeta)$.

VII. Experimental Results

We evaluate the performance of the proposed approach on two well-known datasets: CITESEER [29] and CORA-ML [30]. The statistics of datasets are shown in Table I.

A two-layer graph convolutional network (GCN) [7] is used to perform the sparsification and evaluate the performance of the sparsified graph for node classification. Followed from [15], the hidden size of GCN is 16. To compute the accuracy, the classifiers are trained for 100 iterations and all

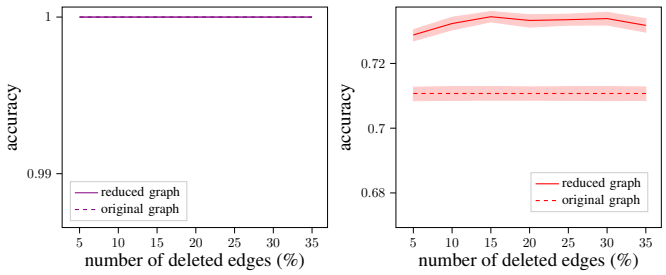

Fig. 1. Accuracy when performing with $\mathcal{L}_{\text{train}}$ on CITESEER with #labeled nodes : #unlabeled nodes = 1 : 9. Left: train-set; Right: test-set.

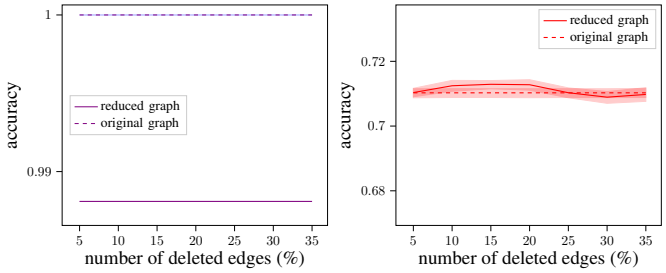

Fig. 2. Accuracy when performing with $\mathcal{L}_{\text{self}}$ on CITESEER with #labeled nodes : #unlabeled nodes = 1 : 9. Left: train-set; Right: test-set.

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 $\nabla_{\hat{A}}\mathcal{L}_{\text{sps}}(f_{\theta_T}(\hat{A}, X))$ by using gradient descent with momentum for $T = 50$ iterations.

We compare the proposed algorithm with the following conventional sparsification techniques: Local Degree [22]; Jaccard Similarity [25]; Forest Fire [22], [23]; Simmelian backbones [26] and SCAN [24]. We use the implementations of those methods in NetworKit [31]. In addition, we show the results of random deletion.

In Section III, $\mathcal{L}_{\text{train}}$ and $\mathcal{L}_{\text{self}}$ are proposed to approximate $\mathcal{L}_{\text{sps}}$. We compare the performance using the two approximations, along with results of the conventional algorithms.

Results on CITESEER: The results on CITESEER dataset are shown in Fig. 1 and Fig. 2. Surprisingly, using $\mathcal{L}_{\text{train}}$ in Fig. 1 significantly improves the test accuracy, even 30% edges are deleted. The training accuracy in the left panel of Fig. 1 shows that the classifier might overfit when using the original graph. The performance of our approach using $\mathcal{L}_{\text{train}}$ is much better than all other methods, shown in Fig. 3.

Results on CORA-ML: The results on CORA-ML dataset are shown in Fig. 4 and Fig. 5. Using $\mathcal{L}_{\text{self}}$ in Fig. 4 the edge deletion first improves the test accuracy, and then the accuracy

Dataset	N_{LCC}	E_{LCC}	Density	Avg Degree	Max Degree	D	K
CORA-ML	2,810	7,981	0.4	11.36	492	2,879	7
CITESEER	2,110	3,668	0.2	5.22	198	3,703	6

TABLE I

DATASET STATISTICS. THE LARGEST CONNECTED COMPONENT (LCC) IS CONSIDERED. N_{LCC}: NUMBER OF NODES; E_{LCC}: NUMBER OF EDGES.

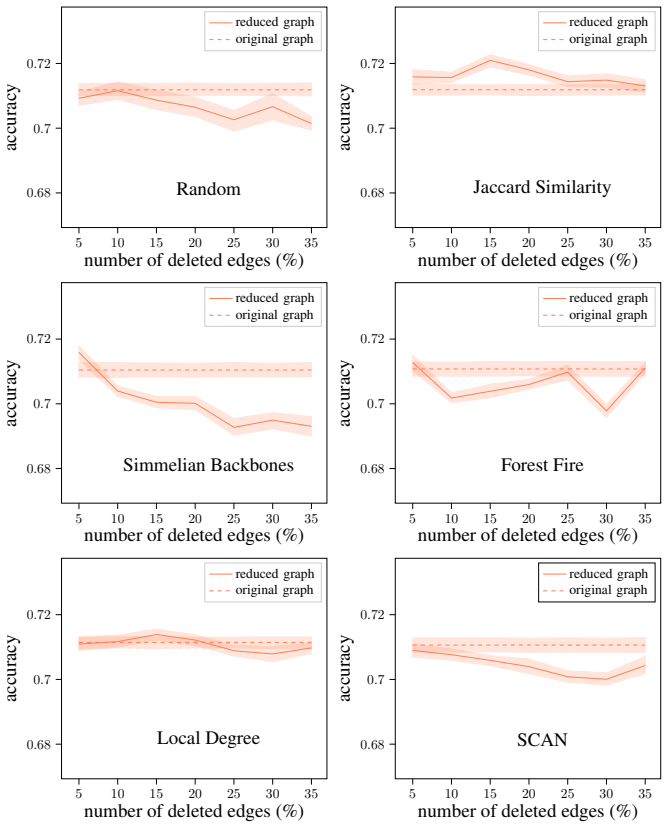

Fig. 3. Accuracy of other algorithms on test-set. CITESEER dataset.

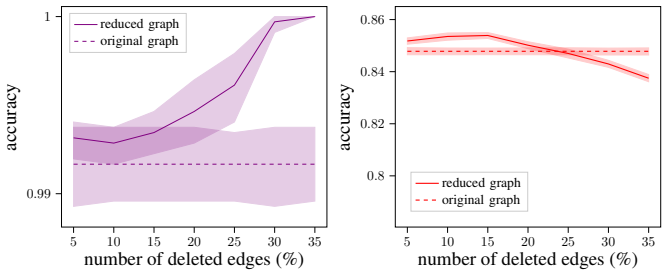

Fig. 4. Accuracy when performing with $\mathcal{L}_{\text{self}}$ on CORA-ML with #labeled nodes : #unlabeled nodes = 1 : 9. Left: train-set; Right: test-set.

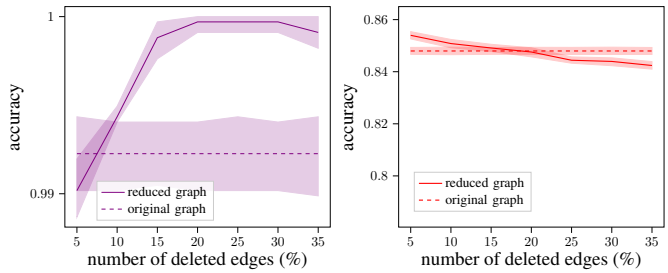

Fig. 5. Accuracy when performing with $\mathcal{L}_{\text{train}}$ on CORA-ML with #labeled nodes : #unlabeled nodes = 1 : 9. Left: train-set; Right: test-set.

decreases as more edges are deleted. The training accuracy in left panel of Fig. 4 shows that the initial edge deletion helps the classifier to fit the data. The result shows that about 20%

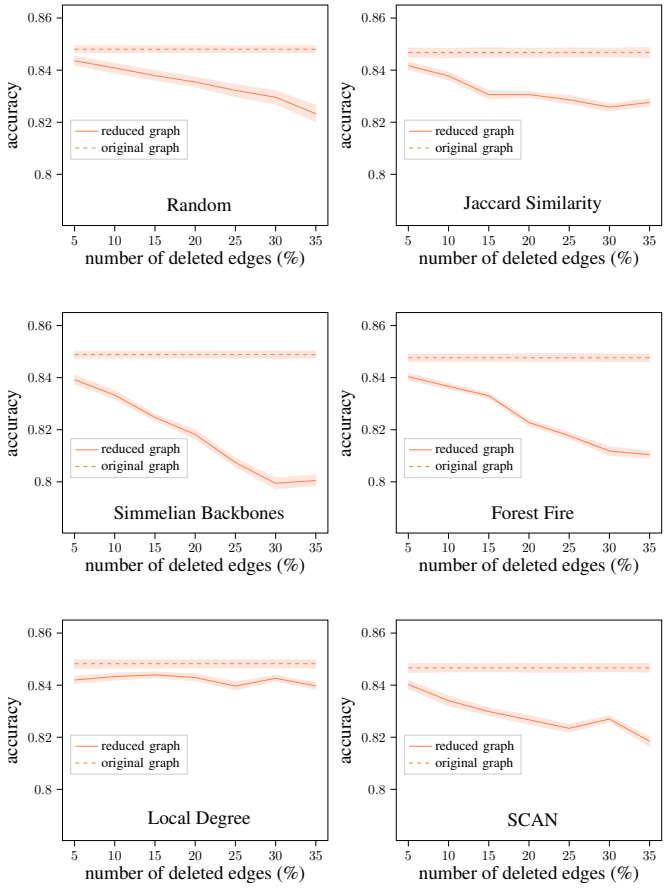

Fig. 6. Accuracy of other algorithms on test-set. CORA-ML dataset.

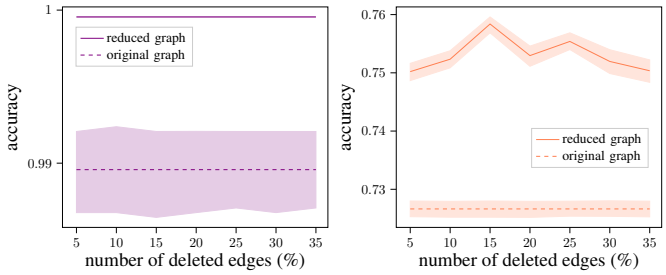

Fig. 7. Accuracy when performing with $\mathcal{L}_{\text{train}}$ on CITESEER with #labeled nodes : #unlabeled nodes = 2 : 8. Left: train-set; Right: test-set.

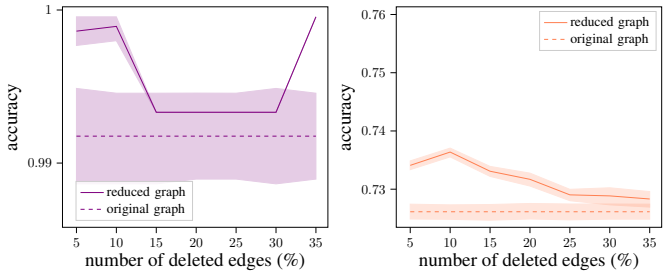

Fig. 8. Accuracy when performing with $\mathcal{L}_{\text{self}}$ on CITESEER with #labeled nodes : #unlabeled nodes = 2 : 8. Left: train-set; Right: test-set.

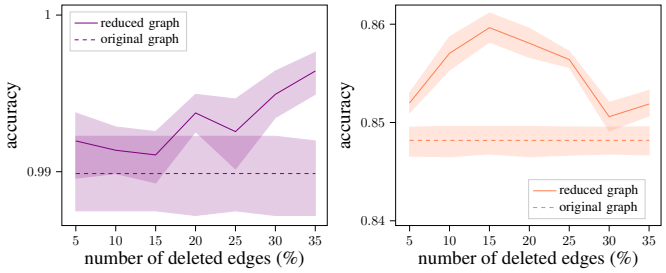

Fig. 9. Accuracy when performing with $\mathcal{L}_{\text{self}}$ on CORA-ML with #labeled nodes : #unlabeled nodes = 2 : 8. Left: train-set; Right: test-set.

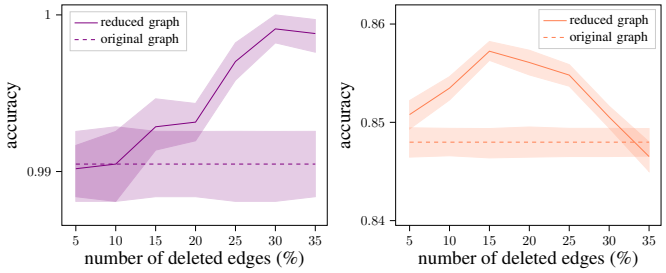

Fig. 10. Accuracy when performing with $\mathcal{L}_{\text{train}}$ on CORA-ML with #labeled nodes : #unlabeled nodes = 2 : 8. Left: train-set; Right: test-set.

edges can be eliminated without hurting the test accuracy. Compared with the performance of other algorithms shown in Fig. 6, our method is clearly superior.

We also show the results when the graphs are split into 20% labeled nodes and 80% unlabeled nodes. The results on CITESEER dataset are shown in Fig. 7 and Fig. 8. The results on CORA-ML dataset are shown in Fig. 9 and Fig. 10. In all cases, 30% edges can be deleted without hurting the test accuracy, and remarkably with the resulting sparser graph, the test accuracy is significantly improved. From the training accuracy in the left panels in Fig. 7-10, the deletion of edges helps the classifiers to fit data.

VIII. Discussion and Future Work

Graph sparsification is an important problem in graph analysis. It helps to understand the information propagation, and to avoid saving unnecessary edges when we only care the node classification setting. The idea of using meta-gradients for edge sparsification for graphs appears to be novel. The experimental results are very encouraging and interesting.

We need to investigate deeply by experimenting on more different datasets, and evaluate the transferability of our technique by using various gradient descent models. Further analysis on properties of the sparsified graphs is also needed. The proposed approach can most likely be applied in related situations, like feature selection or graph classification, that are not explicitly discussed in this paper.

Acknowledgment

I would like to express my sincere gratitude to all people who involve in this work. In particular, I would like to thank my classmate, Harsha Kokel, for the critical suggestions.

REFERENCES

[1] W. Fan, Y. Ma, Q. Li, J. Wang, G. Cai, J. Tang, and D. Yin, "A graph neural network framework for social recommendations," *IEEE Transactions on Knowledge and Data Engineering*, 2020.

[2] M. Wang, Y. Lin, G. Lin, K. Yang, and X.-m. Wu, "M2grl: A multi-task multi-view graph representation learning framework for web-scale recommender systems," in *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, 2020, pp. 2349–2358.

[3] V. Garcia and J. Bruna, "Few-shot learning with graph neural networks," *arXiv preprint arXiv:1711.04043*, 2017.

[4] D. Marcheggiani and I. Titov, "Encoding sentences with graph convolutional networks for semantic role labeling," in *EMNLP 2017-Conference on Empirical Methods in Natural Language Processing, Proceedings*, 2017, pp. 1506–1515.

[5] S. Zhang, H. Tong, J. Xu, and R. Maciejewski, "Graph convolutional networks: Algorithms, applications and open challenges," in *International Conference on Computational Social Networks*. Springer, 2018.

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

[7] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," in *International Conference on Learning Representations, ICLR 2017, Toulon, France*, 2017.

[8] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Liò, and Y. Bengio, "Graph attention networks," in *International Conference on Learning Representations*, 2018.

[9] K. Li, Y. Feng, Y. Gao, and J. Qiu, "Hierarchical graph attention networks for semi-supervised node classification," *Appl. Intell.*, vol. 50, no. 10, pp. 3441–3451, 2020.

[10] D. A. Spielman and N. Srivastava, "Graph sparsification by effective resistances," *SIAM Journal on Computing*, vol. 40, 2011.

[11] D. A. Spielman and S.-H. Teng, "Spectral sparsification of graphs," *SIAM Journal on Computing*, vol. 40, no. 4, pp. 981–1025, 2011.

[12] W.-S. Fung, R. Hariharan, N. J. Harvey, and D. Panigrahi, "A general framework for graph sparsification," *SIAM Journal on Computing*, vol. 48, no. 4, pp. 1196–1223, 2019.

[13] C. Finn, P. Abbeel, and S. Levine, "Model-agnostic meta-learning for fast adaptation of deep networks," in *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, 2017.

[14] A. Nichol, J. Achiam, and J. Schulman, "On first-order meta-learning algorithms," *arXiv preprint arXiv:1803.02999*, 2018.

[15] D. Zügner and S. Günnemann, "Adversarial attacks on graph neural networks via meta learning," in *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA*, 2019.

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

[17] Y. Chen, A. L. Friesen, F. Behbahani, A. Doucet, D. Budden, M. W. Hoffman, and N. de Freitas, "Modular meta-learning with shrinkage," *arXiv preprint arXiv:1909.05557*, 2019.

[18] R. Sheikhpour, M. A. Sarram, S. Gharaghani, and M. A. Z. Chahooki, "A survey on semi-supervised feature selection methods," *Pattern Recognition*, vol. 64, pp. 141–158, 2017.

[19] X. J. Zhu, "Semi-supervised learning literature survey," University of Wisconsin-Madison Department of Computer Sciences, Tech. Rep., 2005.

[20] X. Zhu and A. B. Goldberg, "Introduction to semi-supervised learning," *Synthesis lectures on artificial intelligence and machine learning*, vol. 3, no. 1, pp. 1–130, 2009.

[21] C. Rosenberg, M. Hebert, and H. Schneiderman, "Semi-supervised self-training of object detection models." *WACV/MOTION*, vol. 2, 2005.

[22] M. Hamann, G. Lindner, H. Meyerhenke, C. L. Staudt, and D. Wagner, "Structure-preserving sparsification methods for social networks," *Social Network Analysis and Mining*, vol. 6, no. 1, p. 22, 2016.

[23] J. Leskovec and C. Faloutsos, "Sampling from large graphs," in *Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*, 2006, pp. 631–636.

[24] X. Xu, N. Yuruk, Z. Feng, and T. A. Schweiger, "Scan: a structural clustering algorithm for networks," in *Proceedings of the 13th ACM SIGKDD international conference on Knowledge discovery and data mining*, 2007, pp. 824–833.

[25] V. Satuluri, S. Parthasarathy, and Y. Ruan, "Local graph sparsification for scalable clustering," in *Proceedings of the 2011 ACM SIGMOD International Conference on Management of data*, 2011, pp. 721–732.

[26] B. Nick, C. Lee, P. Cunningham, and U. Brandes, "Simmelian backbones: Amplifying hidden homophily in facebook networks," in *Proceedings of the 2013 IEEE/ACM international conference on advances in social networks analysis and mining*, 2013, pp. 525–532.

[27] Y. Bengio, "Gradient-based optimization of hyperparameters," *Neural computation*, vol. 12, no. 8, pp. 1889–1900, 2000.

[28] W. L. Hamilton, R. Ying, and J. Leskovec, "Inductive representation learning on large graphs," in *Proceedings of the 31st International Conference on Neural Information Processing Systems*, 2017, pp. 1025–1035.

[29] P. Sen, G. Namata, M. Bilgic, L. Getoor, B. Galligher, and T. Eliassi-Rad, "Collective classification in network data," *AI magazine*, vol. 29, no. 3, pp. 93–93, 2008.

[30] A. K. McCallum, K. Nigam, J. Rennie, and K. Seymore, "Automating the construction of internet portals with machine learning," *Information Retrieval*, vol. 3, no. 2, pp. 127–163, 2000.

[31] C. L. Staudt, A. Sazonovs, and H. Meyerhenke, "Networkit: A tool suite for large-scale complex network analysis," *Network Science*, vol. 4, no. 4, pp. 508–530, 2016.