Graph Sparsification via Meta-Learning

Guihong Wan and Harsha Kokel

Department of Computer Science
The University of Texas at Dallas
Contributions

An edge sparsification algorithm for undirected graphs

Delete edges \[\rightarrow\] Preserve node classification accuracy

• Formulate as a bi-level optimization problem

\[
\hat{G}^* = \min_{\hat{G} \in \Phi(G)} \mathcal{L}_{sps}(f_{\theta^*}(\hat{G}), Y_U)
\]

\[s.t. \quad \theta^* = \arg\min_{\theta} \mathcal{L}_{train}(f_{\theta}(\hat{G}), Y_L)\]

• Use meta-gradients to solve it.

\[
\nabla^\text{meta} \hat{A} := \nabla_{\hat{A}} \mathcal{L}_{sps}(f_{\theta^*}(\hat{A}, X), Y_U),
\]

\[s.t. \quad \theta^* = \arg\min_{\theta} \mathcal{L}_{train}(f_{\theta}(\hat{A}, X), Y_L)\]
Outline

• Introduction
  • Graph Sparsification
  • Semi-Supervised Node Classification

• Our Approach
  • Modeling the Problem
  • Meta-Gradients
  • Score Matrix
  • The Proposed Algorithm

• Experimental Results
Introduction

• Graph sparsification
• Semi-supervised node classification
Graph Sparsification

- **Edge Sparsification:**
  reduce the edges of a graph while preserving structural / statistical properties of interest.

- **Density** for undirected graphs:
  \[
  \frac{2M}{N(N - 1)} \]
  number of edges
  number of nodes

We focus on edge sparsification while preserving the node classification accuracy.
Semi-Supervised Node Classification

• **Graph:** $G = (A, X), Y_L, Y_U$
  - adjacency matrix
  - attribute matrix
  - labels
  - to be predicted

• **Task:** Node classification
  - Input: $(A, X, Y_L)$
  - Output: predicted $Y_U$

• **Example:** Graph Convolution Network (GCN)
Two-layer GCN:

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

The \(l^{th}\) GCN Layer:

\[
H_{l+1} = \sigma(\tilde{D}^{-\frac{1}{2}} \tilde{A} \tilde{D}^{-\frac{1}{2}} H_l W_l)
\]

\(H_1 = X\), otherwise the output of previous layer.

\[
\tilde{A} = A + I_N, \quad \tilde{D} = \sum_j \tilde{A}_{ij}
\]

\(W\): what to learn.
Understand intuitively:

\[ H_2 = \sigma(\tilde{A}XW_1) \] when no normalization.

**Main idea:** learn a node \( v \)'s representation by aggregating its own feature \( x_v \) and its neighbors' feature \( x_u \), for all \( u \in N(v) \).
Understand intuitively:

\[ H_2 = \sigma(\tilde{A}XW_1) \] when no normalization.

\[
\begin{bmatrix}
1 & 1 & 1 & 0 \\
1 & 1 & 1 & 1 \\
1 & 1 & 1 & 0 \\
0 & 1 & 0 & 1
\end{bmatrix}
\begin{bmatrix}
x_{11} & x_{12} & x_{13} & x_{14} \\
x_{21} & x_{22} & x_{23} & x_{24} \\
x_{31} & x_{32} & x_{33} & x_{34} \\
x_{41} & x_{42} & x_{43} & x_{44}
\end{bmatrix} =
\begin{bmatrix}
x_1 \\
x_2 \\
x_3 \\
x_4
\end{bmatrix}
\]

\[
\begin{bmatrix}
x_{11} + x_{21} + x_{31} \\
x_{12} + x_{22} + x_{32} + x_{42} \\
\vdots
\end{bmatrix}
\]

**Main idea:** learn a node \( v \)'s representation by aggregating its own feature \( x_v \) and its neighbors' feature \( x_u \), for all \( u \in N(v) \).
Our Approach

• Modeling the problem
• Meta-gradients
• Score matrix
Modeling the Problem

- **Given**: $G = (A, X)$, labeled nodes: $Y_L$.
- **Goal of semi-supervised node classification**: learn a function $f_\theta$ to map each node to a class.

$$\theta^* = \arg\min_\theta \mathcal{L}_{\text{train}}(f_\theta(G), Y_L)$$
Modeling the problem (cont.)

- **Given:** $G = (A, X)$,
  labeled nodes: $Y_L$,
  number of edges to be deleted: $\zeta$.

- **Goal:** delete edges
  but reduce the loss of node classification accuracy on unlabeled nodes:

$$L_{SPS}(\tilde{Y}_U, Y_U)$$

Predicted labels  True labels of unlabeled nodes
Modeling the problem (cont.)

• Formulate as a bi-level optimization problem

\[
\hat{G}^* = \min_{\hat{G} \in \Phi(G)} \mathcal{L}_{sps}(f_{\theta^*}(\hat{G}), Y_U)
\]

\[
s.t. \quad \theta^* = \arg\min_{\theta} \mathcal{L}_{\text{train}}(f_\theta(\hat{G}), Y_L)
\]

• **Inner optimization**: train the model over labeled nodes for predicating labels of unlabeled nodes.

• **Outer objective**: for the sparsifier which aims to minimize the loss of classification accuracy.

\(Y_U\) is unknown to the sparsifier.
Modeling the problem

\[
\hat{G}^* = \min_{\hat{G} \in \Phi(G)} \mathcal{L}_{sps}(f_{\theta^*}(\hat{G}), Y_U)
\]

\[
s.t. \quad \theta^* = \arg \min_{\theta} \mathcal{L}_{\text{train}}(f_{\theta}(\hat{G}'), Y_L)
\]

unknown to the sparsifier.

Three options to approximate:

• \( L_{sps} \approx L_{\text{train}} \): compute from \( Y_L \).

• \( L_{sps} \approx L_{\text{self}} \): the sparsifier can train a classifier on labeled data to estimate the labels of unlabeled nodes \( \hat{Y}_U \).

• \( L_{sps} \approx L_{\text{both}} \): combine \( Y_L \) and \( \hat{Y}_U \).
Meta-Gradients

- Adjacency matrix $\rightarrow$ hyperparameters
- Compute the gradients of the sparsifier’s loss w.r.t the adjacency matrix

\[
\nabla^\text{meta}_{\hat{A}} := \nabla_{\hat{A}} \mathcal{L}_{\text{sps}}(f_{\theta^*}(\hat{A}, X), Y_U)
\]

\[
s.t. \quad \theta^* = \arg\min_{\theta} \mathcal{L}_{\text{train}}(f_\theta(\hat{A}, X), Y_L)
\]

- Indicate how the sparsifier’s loss $L_{sps}$ will change after training on the simplified graph.
Meta-Gradients (cont.)

- Adjacency matrix → hyperparameters
- Compute the gradients of the sparsifier’s loss w.r.t the adjacency matrix

\[
\nabla_{\hat{A}}^{\text{meta}} := \nabla_{\hat{A}} \mathcal{L}_{\text{sps}}(f_{\theta^*}(\hat{A}, X), Y_U) \\
s.t. \quad \theta^* = \arg \min_{\theta} \mathcal{L}_{\text{train}}(f_{\theta}(\hat{A}, X), Y_L)
\]

- Inner update: \( \theta_{t+1} = \theta_t - \alpha \nabla_{\theta_t} \mathcal{L}_{\text{train}}(f_{\theta_t}(\hat{G}), Y_L) \)

- Outer update: \( \hat{A}^{k+1} = \hat{A}^k - \beta \nabla_{\hat{A}_k}^{\text{meta}}, \text{ with } \hat{A}^0 = A \)
Score Matrix

\[ \hat{A}^{k+1} = \hat{A}^k - \beta \nabla_{\hat{A}^k} \text{meta}, \text{ with } \hat{A}^0 = A \]

- 0/1 problem: an edge is either deleted or kept (A is discrete)
- Score matrix:

\[ S = \nabla_{\hat{A}} \text{meta} \odot \hat{A} \]

\[ e^* = \arg \max_{e(i,j) \in \hat{A}} S(i, j) \]

\[ e(i,j) \in \hat{A} \text{ and } e(i,j) \in \Phi(G) \]
Score Matrix (cont.)

\[ \hat{A}^{k+1} = \hat{A}^k - \beta \nabla_{\hat{A}} \text{meta} \], with \( \hat{A}^0 = A \)

\[ S = \nabla_{\hat{A}} \text{meta} \odot \hat{A} \]

- Deletion: From 1 to 0;
- Positive gradients are preferred.

For weighted graphs, we can learn an indicator matrix initialized as 1 if there is an edge.
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: $\tilde{G}^* = (\tilde{A}^*, X)$

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

- Results on CiteSeer dataset
- Results on Cora-ML dataset
Datasets

<table>
<thead>
<tr>
<th>Dataset</th>
<th>#Nodes</th>
<th>#Edges</th>
<th>Density</th>
<th>Avg degree</th>
<th>Max degree</th>
<th>Test Acc</th>
</tr>
</thead>
<tbody>
<tr>
<td>CiteSeer</td>
<td>2,110</td>
<td>3,668</td>
<td>0.2</td>
<td>5.22</td>
<td>198</td>
<td>0.71</td>
</tr>
<tr>
<td>Cora-ML</td>
<td>2,810</td>
<td>7,981</td>
<td>0.4</td>
<td>11.36</td>
<td>492</td>
<td>0.85</td>
</tr>
</tbody>
</table>

- We only consider the largest connected component.
- 10% labeled nodes for training; 90% unlabeled nodes for testing.
Main Observations

• Our algorithm works better than the conventional methods.
• $L_{\text{train}}$ works better when overfitting;
• $L_{\text{self}}$ works better when underfitting.
CiteSeer

$L_{\text{train}}$

$L_{\text{self}}$

Train-set

Test-set
Comparison

Ours
30% edges deleted
Test Acc : 0.73

Original Graph
Test Acc : 0.71
Ours 30% edges deleted
Test Acc : 0.73

Local Degree 30% edges deleted
Test Acc : 0.71
30% edges deleted
Test Acc : 0.73

Ours

Simmelian

30% edges deleted
Test Acc : 0.69
Cora-ML

$L_{\text{train}}$  

$L_{\text{self}}$  

Train-set

Test-set

accuracy vs. number of deleted edges (%) for reduced and original graphs.
Comparison

Ours
30% edges deleted
Test Acc : 0.843

Original Graph
Test Acc : 0.848
Thank You!