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

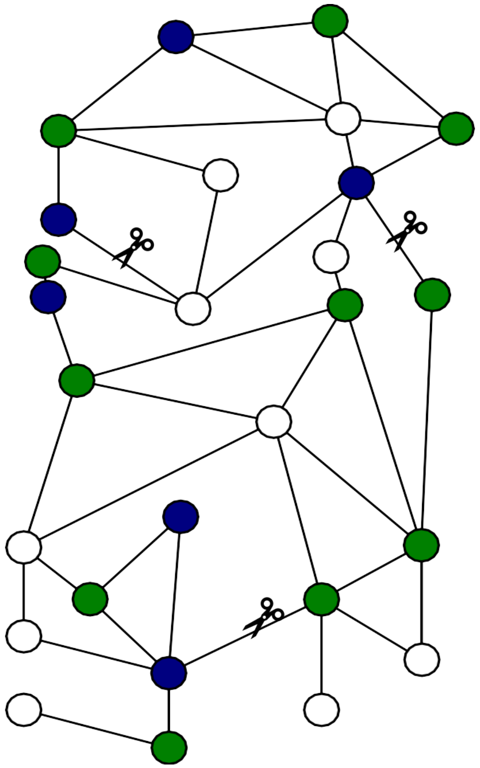
Delete edges $\longleftrightarrow$ Preserve node classification accuracy

- **Formulate as a bi-level optimization problem**

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

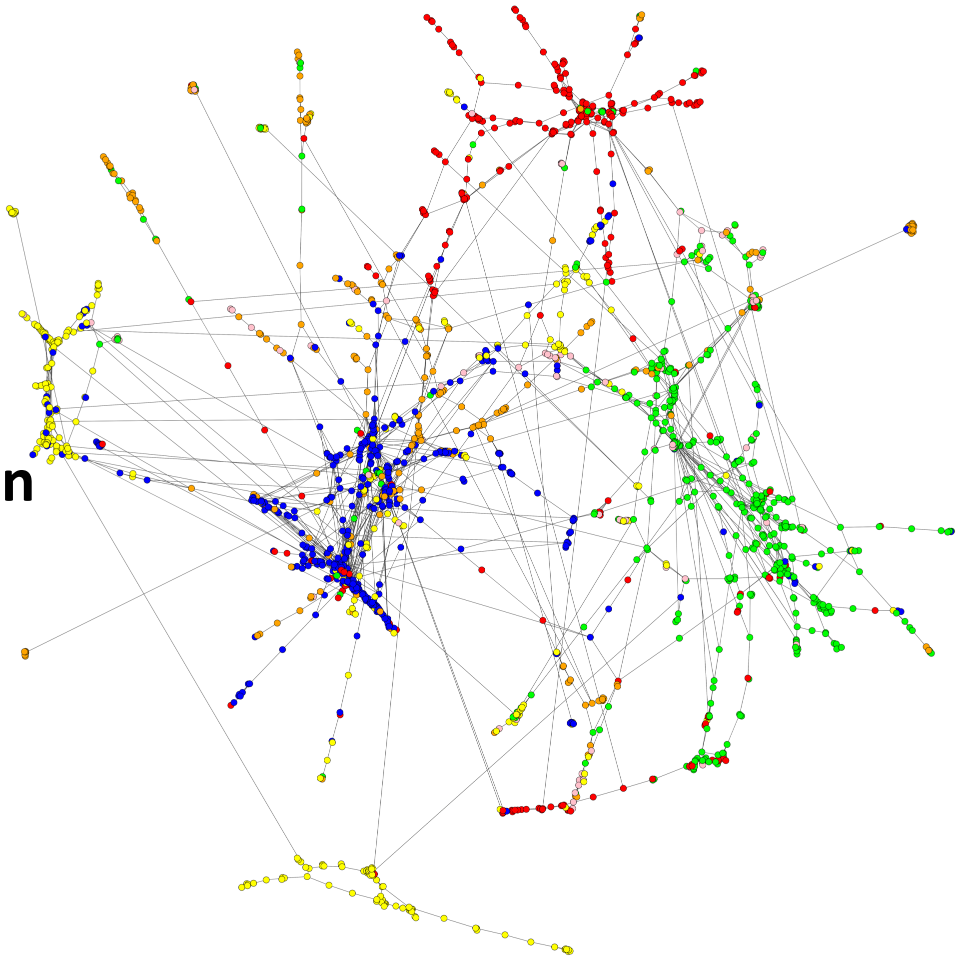
- **Use meta-gradients to solve it.**

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



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:**

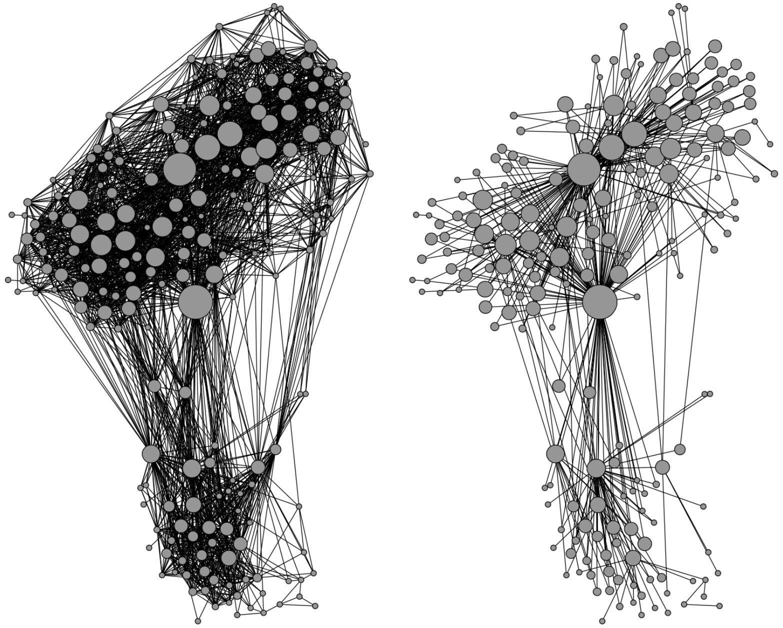
reduce the edges of a graph while preserving structural / statistical properties of interest.

- **Density** for undirected graphs:

$$\frac{2M}{N(N-1)}$$

number of edges

number of nodes



Hamann et al. 2016

We focus on edge sparsificaton while preserving the node classification accuracy.

Semi-Supervised Node Classification

- **Graph:** $G = (A, X), Y_L, Y_U$ ← to be predicted

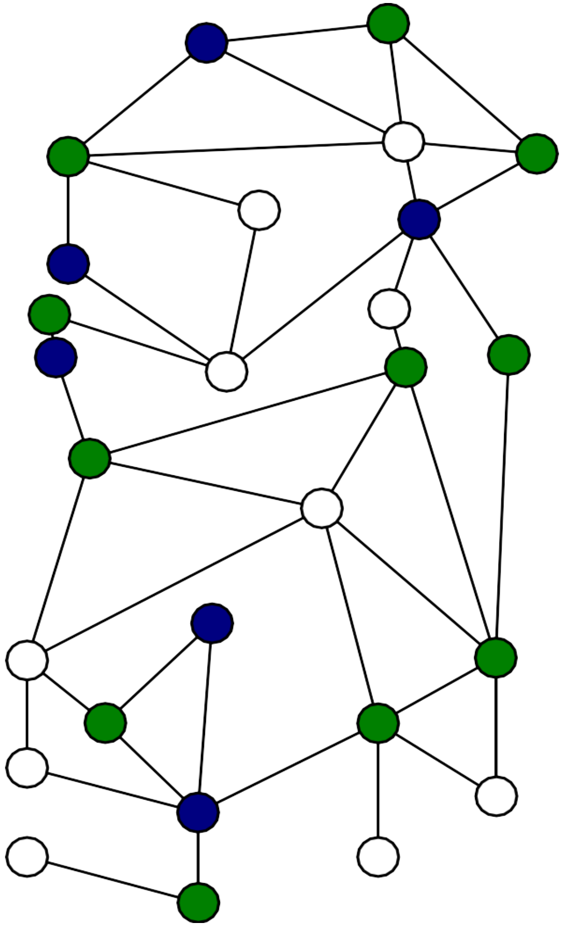
adjacency matrix

attribute matrix

labels

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

- **Example:** Graph Convolution Network (GCN)



GCN

Two-layer GCN:

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

The l^{th} GCN Layer: $H_{l+1} = \sigma(\widetilde{D}^{-\frac{1}{2}}\tilde{A}\widetilde{D}^{-\frac{1}{2}}H_l W_l)$

$\boldsymbol{H_1 = X}$, otherwise the output of previous layer.

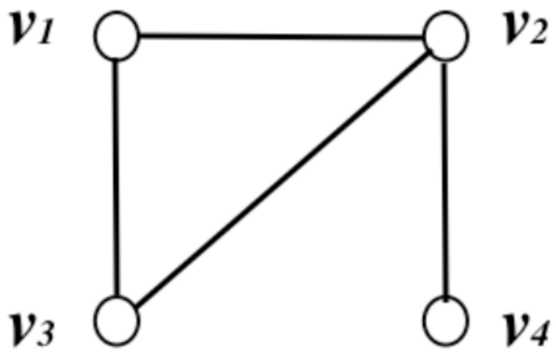
$\widetilde{\boldsymbol{A}} = \boldsymbol{A} + \boldsymbol{I_N}, \qquad \widetilde{\boldsymbol{D}} = \sum_{\boldsymbol{j}} \widetilde{\boldsymbol{A}}_{\boldsymbol{ij}}$

$\boldsymbol{W}$: what to learn.

GCN (cont.)

Understand intuitively:

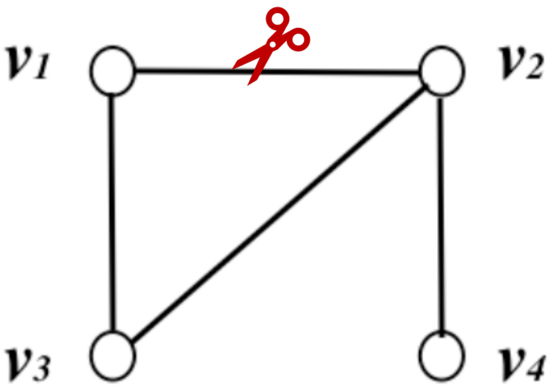
$\boldsymbol{H_2 = \sigma(\tilde{A}XW_1)}$ when no normalization.



$$
\overset{\tilde{A}}{\begin{bmatrix} 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 \\ 0 & 1 & 0 & 1 \end{bmatrix}}
\quad
\begin{array}{cc} & X \\ \begin{bmatrix} x_{11} & x_{12} & x_{13} & x_{14} \\ x_{21} & x_{22} & x_{23} & x_{24} \\ x_{31} & x_{32} & x_{33} & x_{34} \\ x_{41} & x_{42} & x_{43} & x_{44} \end{bmatrix} & \begin{array}{l} \leftarrow x_1 \\ \leftarrow x_2 \\ \leftarrow x_3 \\ \leftarrow x_4 \end{array} \end{array}
\qquad
\overset{AX}{\begin{bmatrix} x_{11} + x_{21} + x_{31} & x_{12} + x_{22} + x_{32} & \cdots \\ & & \\ & & \\ & & \end{bmatrix}}
$$

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

GCN (cont.)



Understand intuitively:

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

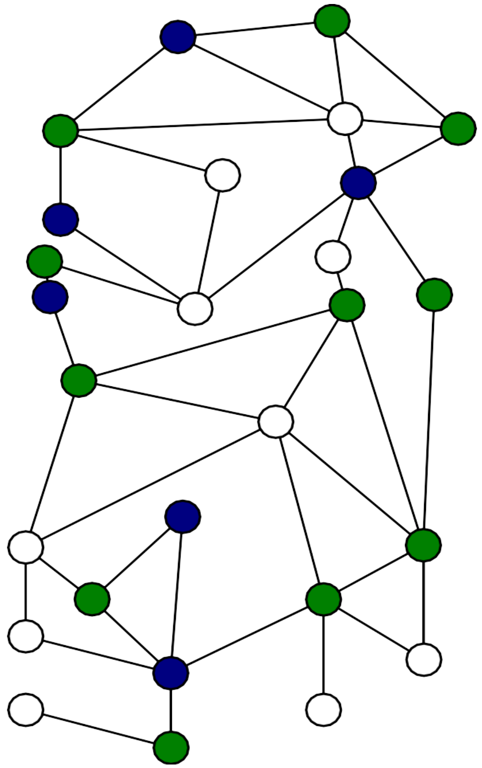
$$\underset{\tilde{A}}{\begin{bmatrix} 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 \\ 0 & 1 & 0 & 1 \end{bmatrix}} \quad \underset{X}{\begin{bmatrix} x_{11} & x_{12} & x_{13} & x_{14} \\ x_{21} & x_{22} & x_{23} & x_{24} \\ x_{31} & x_{32} & x_{33} & x_{34} \\ x_{41} & x_{42} & x_{43} & x_{44} \end{bmatrix}} \begin{array}{l} \leftarrow x_1 \\ \leftarrow x_2 \\ \leftarrow x_3 \\ \leftarrow x_4 \end{array} \qquad \underset{AX}{\begin{bmatrix} x_{11} + x_{21} + x_{31} & x_{12} + x_{22} + x_{32} & \cdots \\ & & \\ & & \\ & & \end{bmatrix}}$$

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 $\boldsymbol{f_\theta}$ to map each node to a class.

$$\boxed{\theta^* = \arg\min_{\theta} \mathcal{L}_{\text{train}}\left(f_\theta(G), Y_L\right)}$$

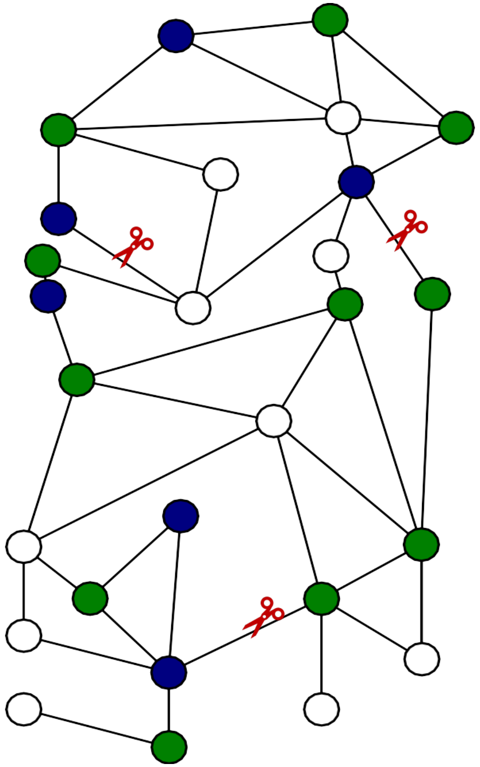
Modeling the problem (cont.)

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

- **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**

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

Outer

Inner

- 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

$$\hat{G}^* = \min_{\hat{G} \epsilon \Phi(G)} \mathcal{L}_{\text{sps}}(f_{\theta^*}(\hat{G}), \boxed{Y_U})$$

unknown to the sparsifier.

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

Three options to approximate:

- $L_{sps} \approx L_{train}$: compute from Y_L.
- $L_{sps} \approx L_{self}$: the sparsifier can train a classifier on labeled data to estimate the labels of unlabeled nodes $\hat{Y}_U$.
- $L_{sps} \approx L_{both}$: combine Y_L and $\hat{Y}_U$.

Meta-Gradients

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

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

- 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

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

- Inner update: $\theta_{t+1} = \theta_t - \alpha \nabla_{\theta_t} \mathcal{L}_{\text{train}}(f_{\theta_t}(\hat{G}), Y_L)$
- Outer update: $\hat{A}^{k+1} = \hat{A}^k - \beta \nabla^{\text{meta}}_{\hat{A}^k}, \text{ with } \hat{A}^0 = A$

Score Matrix

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

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

$$\boxed{S = \nabla^{\text{meta}}_{\hat{A}} \odot \hat{A}}$$

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

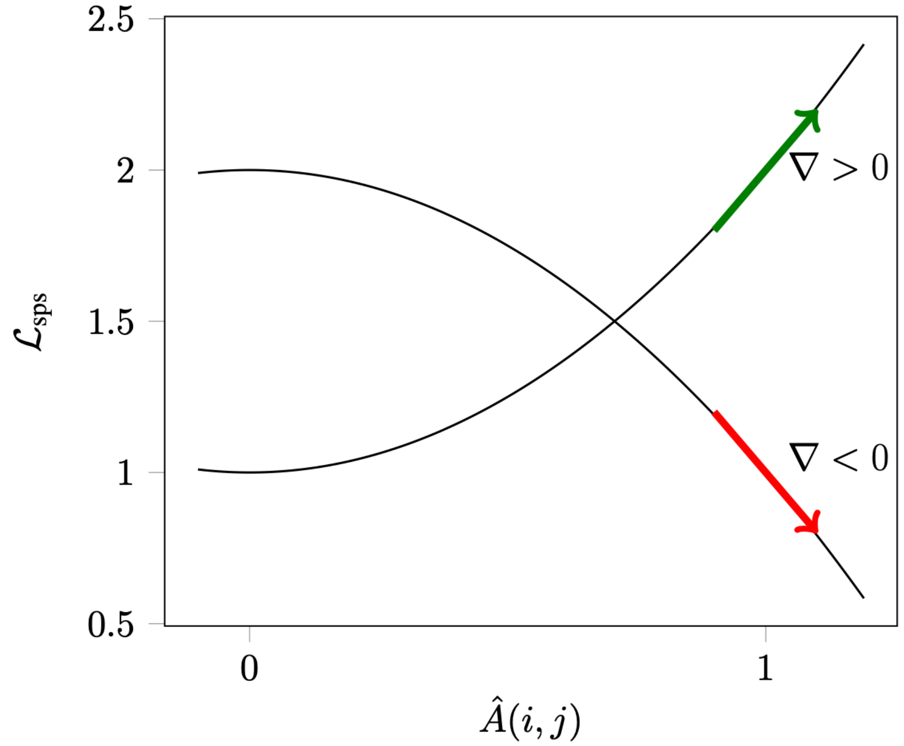
Score Matrix (cont.)

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

$$\boxed{S = \nabla^{\text{meta}}_{\hat{A}} \odot \hat{A}}$$

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

Figure 1. Illustration of the score matrix.



For weighted graphs, we can learn an indicator matrix initialized as 1 if there is an edge.

Algorithm

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: $\hat{G}^* = (\hat{A}^*, X)$

$\hat{Y}_U \leftarrow$ estimated labels of unlabeled nodes using self-training;
$\hat{A} \leftarrow A$;
while $\zeta > 0$ **do**
  $\theta_0 \leftarrow$ initialize randomly;
  for t in $0 \dots T-1$ **do**
    $\theta_{t+1} = \theta_t - \alpha \nabla_{\theta_t} \mathcal{L}_{\text{train}}(f_{\theta_t}(\hat{A}, X), Y_L)$;
  end for
  $\nabla_{\hat{A}}^{\text{meta}} \leftarrow \nabla_{\hat{A}} \mathcal{L}_{\text{self}}(f_{\theta_T}(\hat{A}, X), \hat{Y}_U)$;
  $S = \nabla_{\hat{A}}^{\text{meta}} \odot \hat{A}$;
  $e^* \leftarrow$ the maximum entry (i, j) in $S(i, j)$ that satisfies the constraints $\Phi(G)$;
  $\hat{A} \leftarrow$ remove edge e^*;
  $\zeta \mathrel{-}= 1$;
end while
$\hat{G}^* \leftarrow (\hat{A}, X)$;
return $\hat{G}^*$.

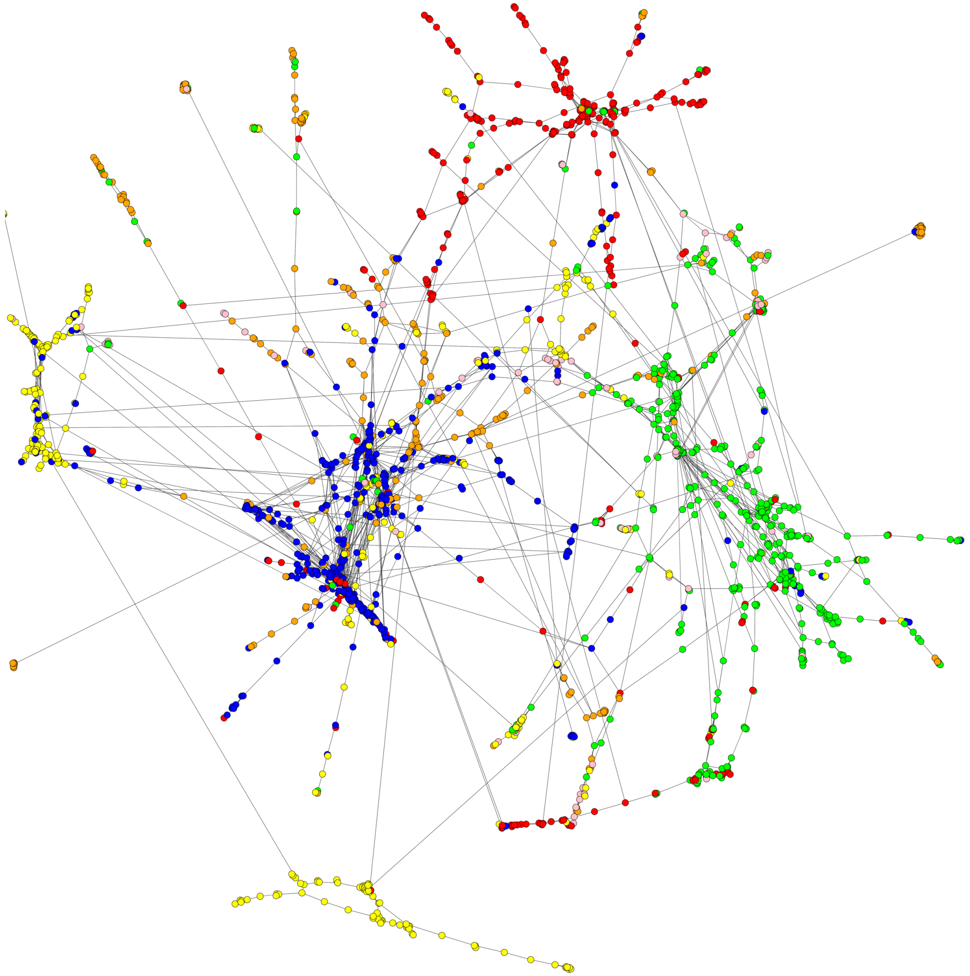
Experimental Results

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

Datasets

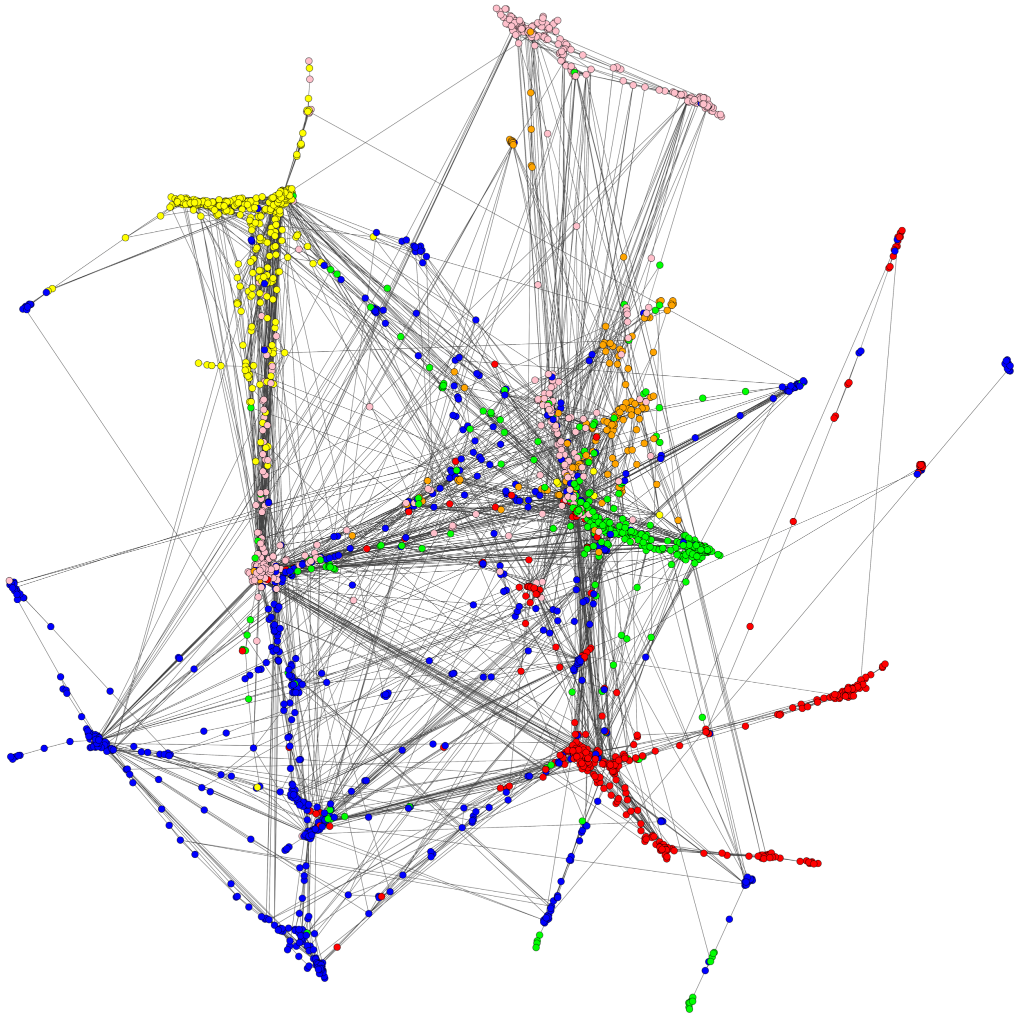
	#Nodes	#Edges	Density	Avg degree	Max degree	Test Acc
CiteSeer	2,110	3,668	0.2	5.22	198	0.71
Cora-ML	2,810	7,981	0.4	11.36	492	0.85

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



CiteSeer

Density: 0.2



Cora-ML

Density: 0.4

Main Observations

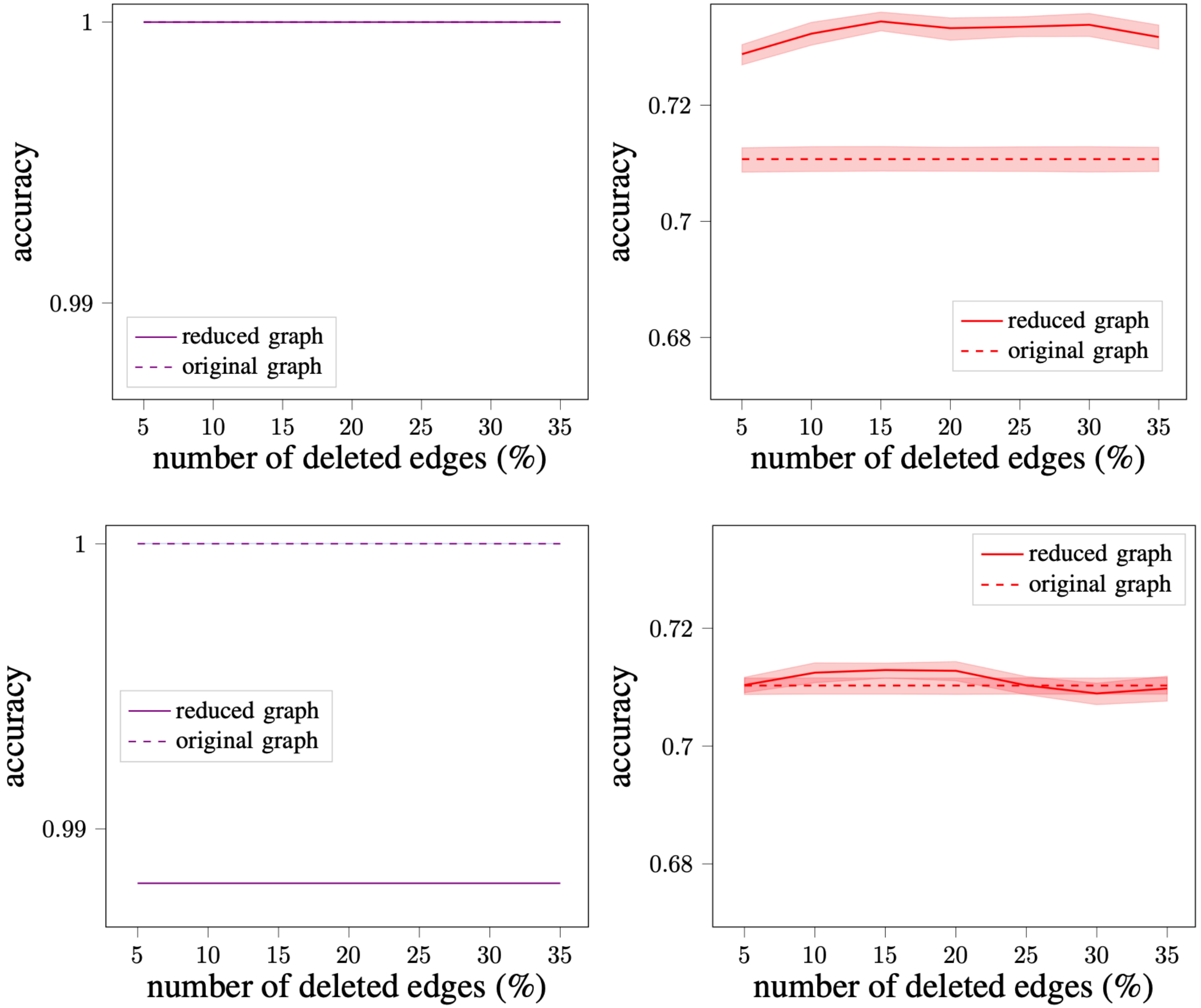
- Our algorithm works better than the conventional methods.
- L_{train} works better when overfitting;
- L_{self} works better when underfitting.

CiteSeer

Train-set   Test-set

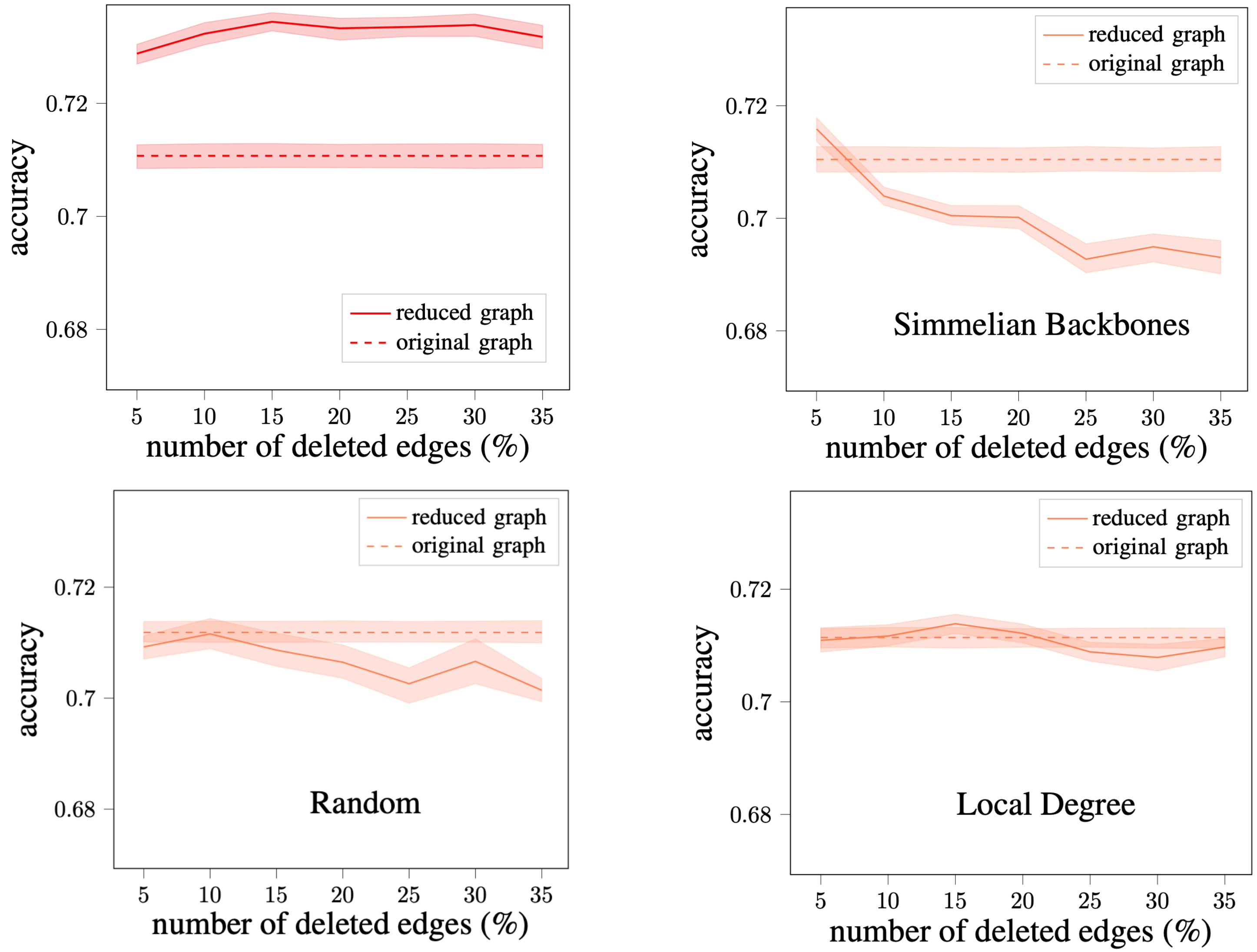
L_{train} →

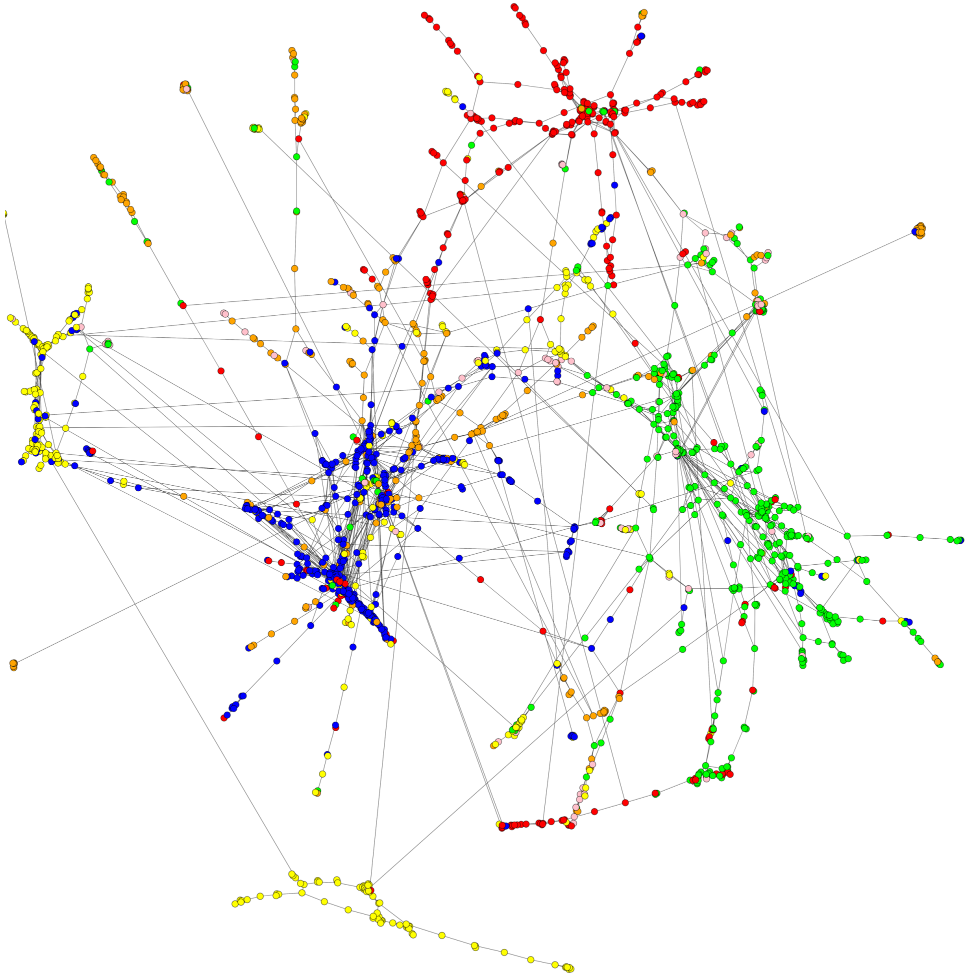
L_{self} →



Comparison

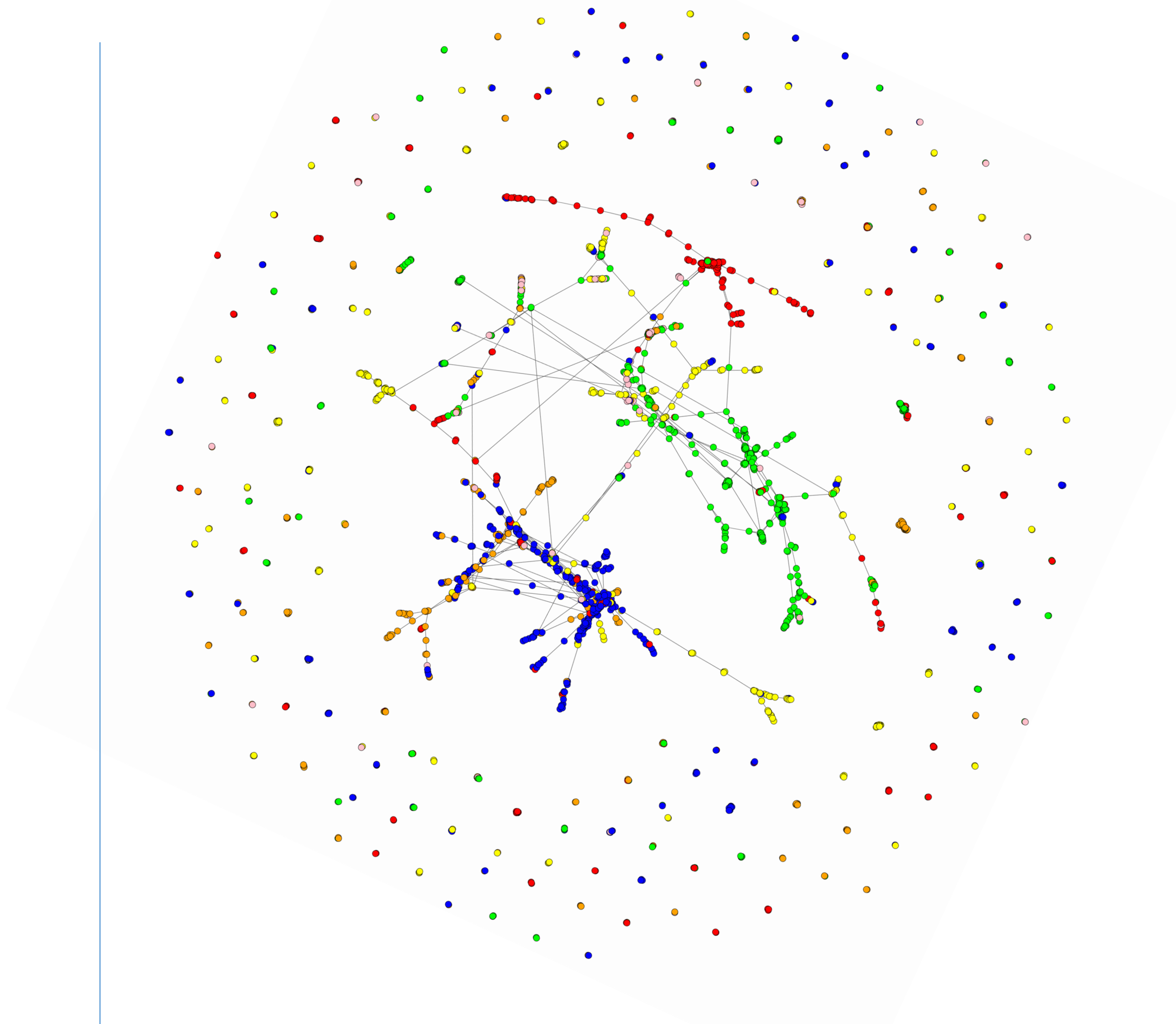
Ours





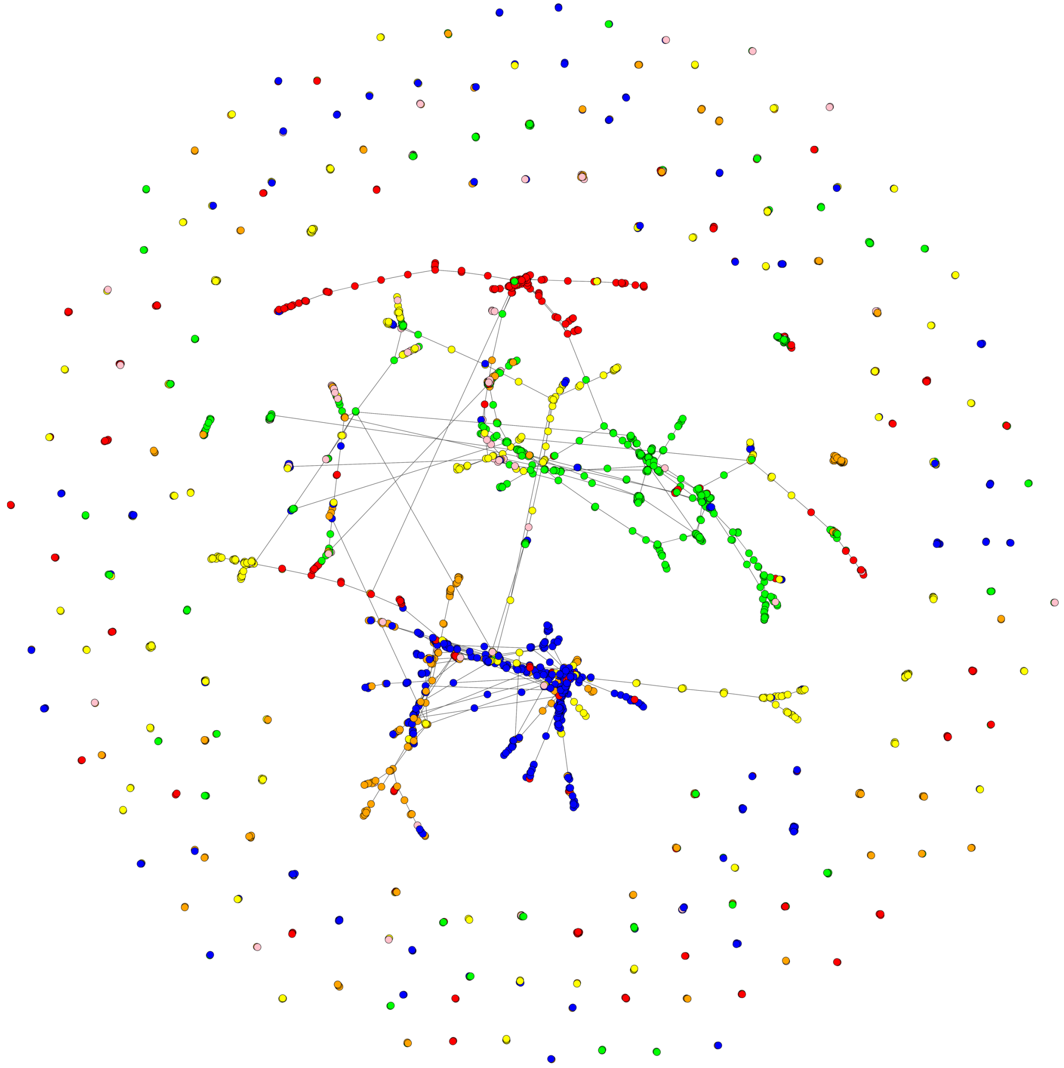
Original Graph

Test Acc : 0.71



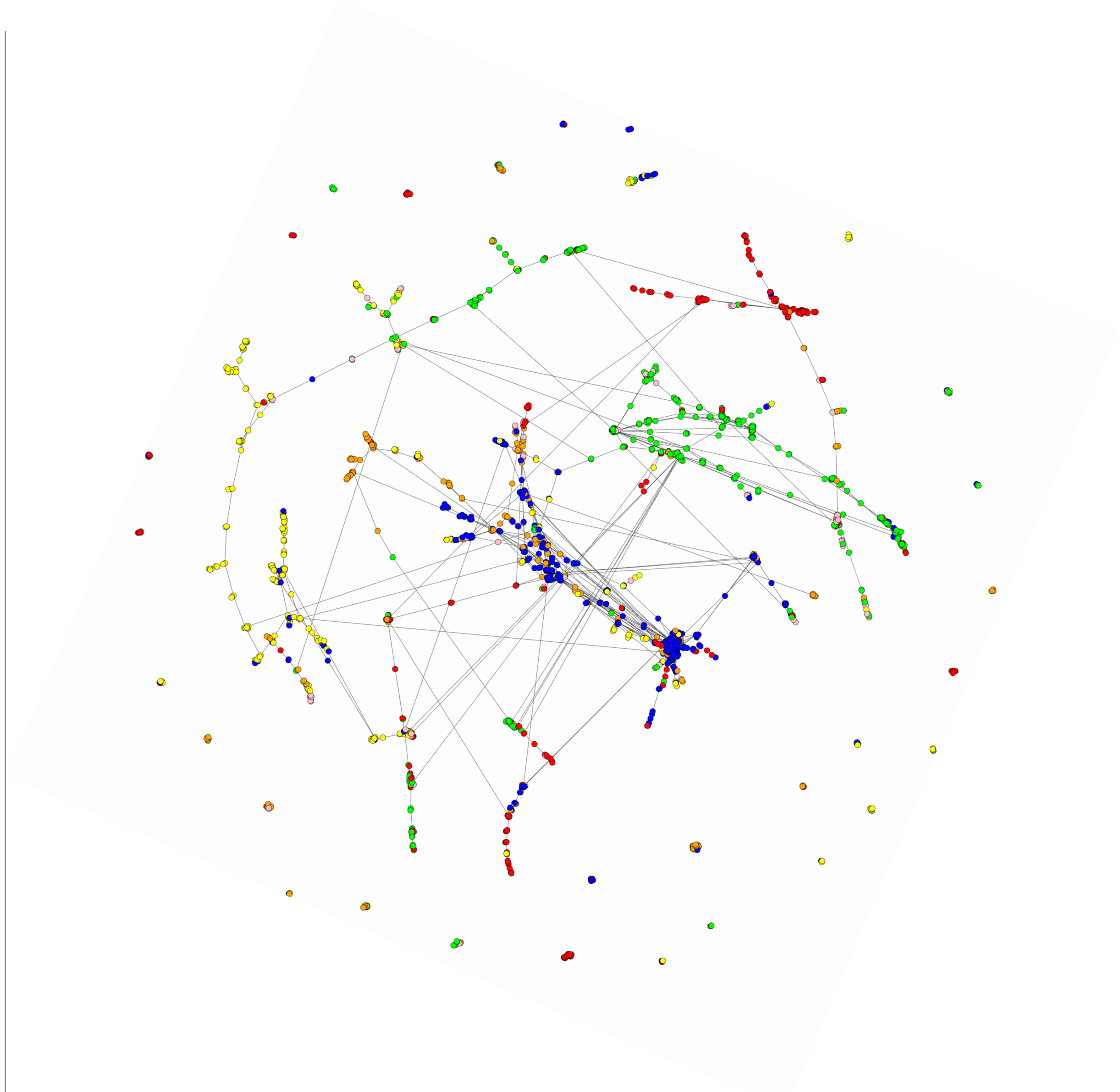
30% edges deleted

Test Acc : 0.73



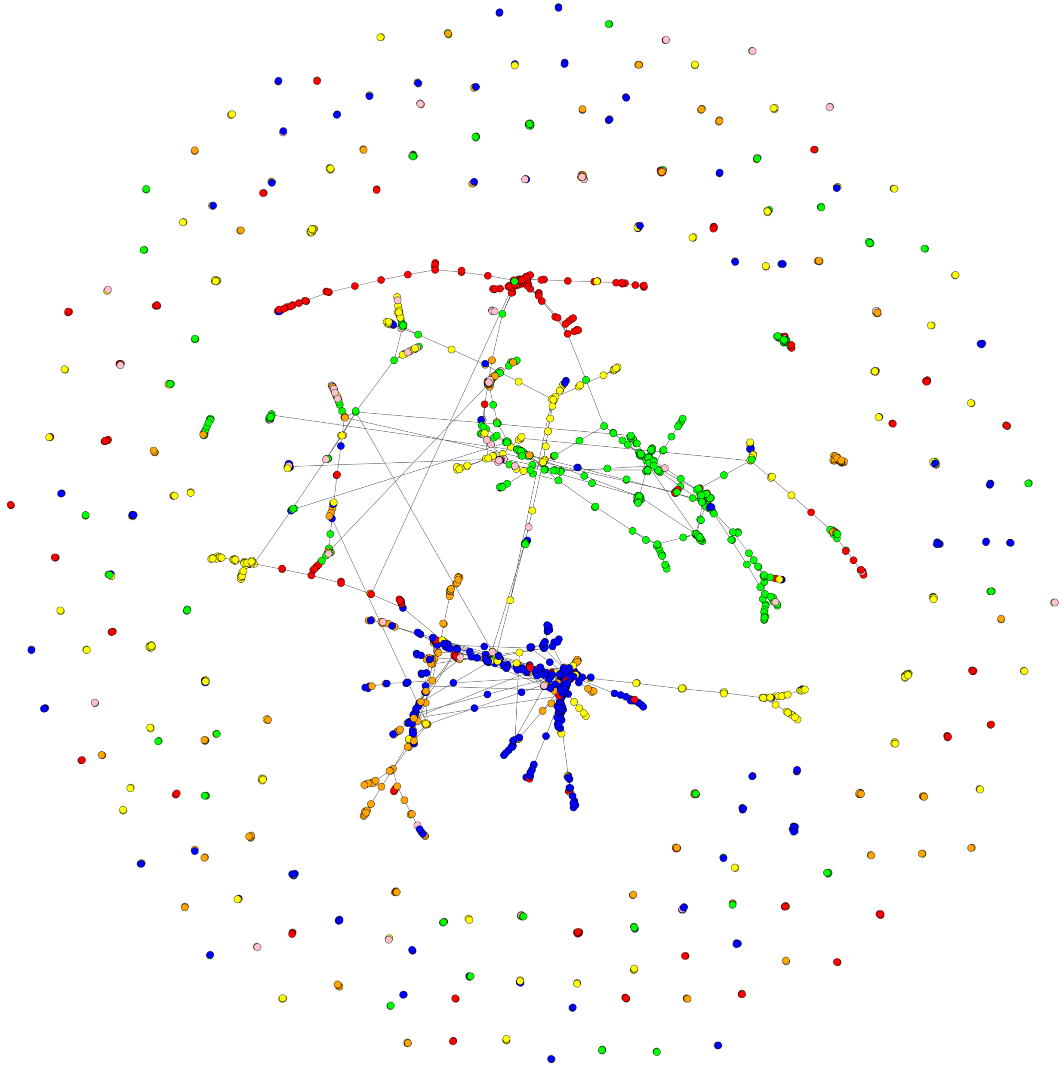
Ours

30% edges deleted
Test Acc : 0.73



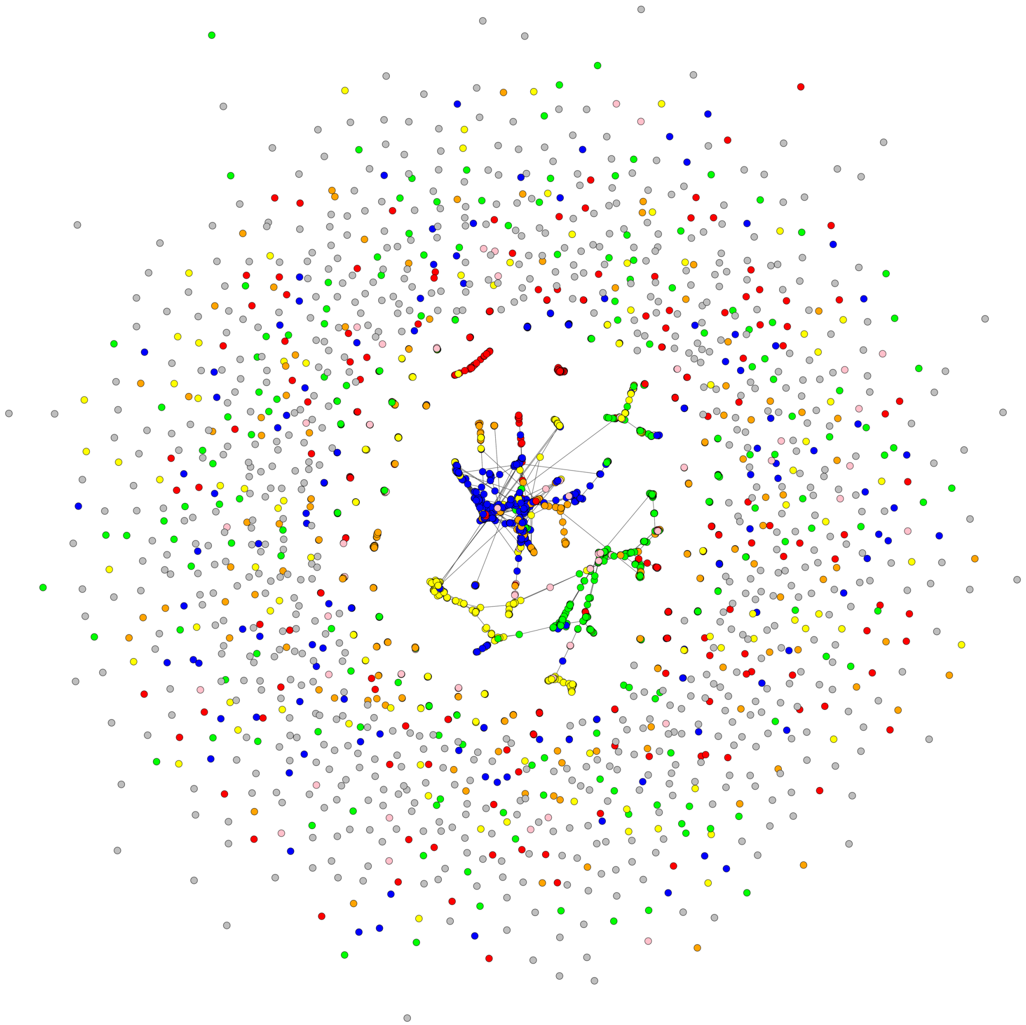
Local Degree

30% edges deleted
Test Acc : 0.71



Ours

30% edges deleted
Test Acc : 0.73



Simmelian

30% edges deleted
Test Acc : 0.69

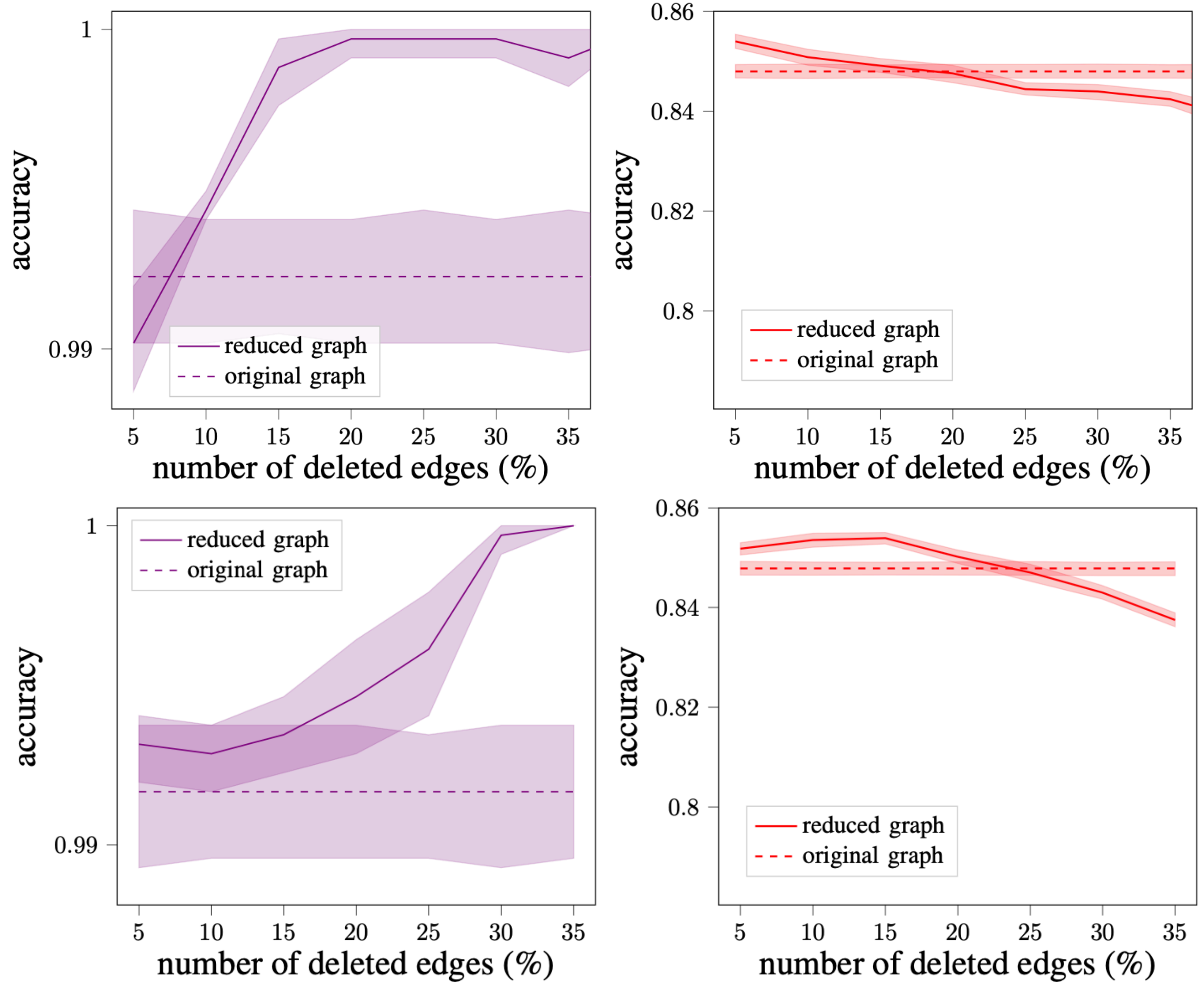
Cora-ML

Train-set

Test-set

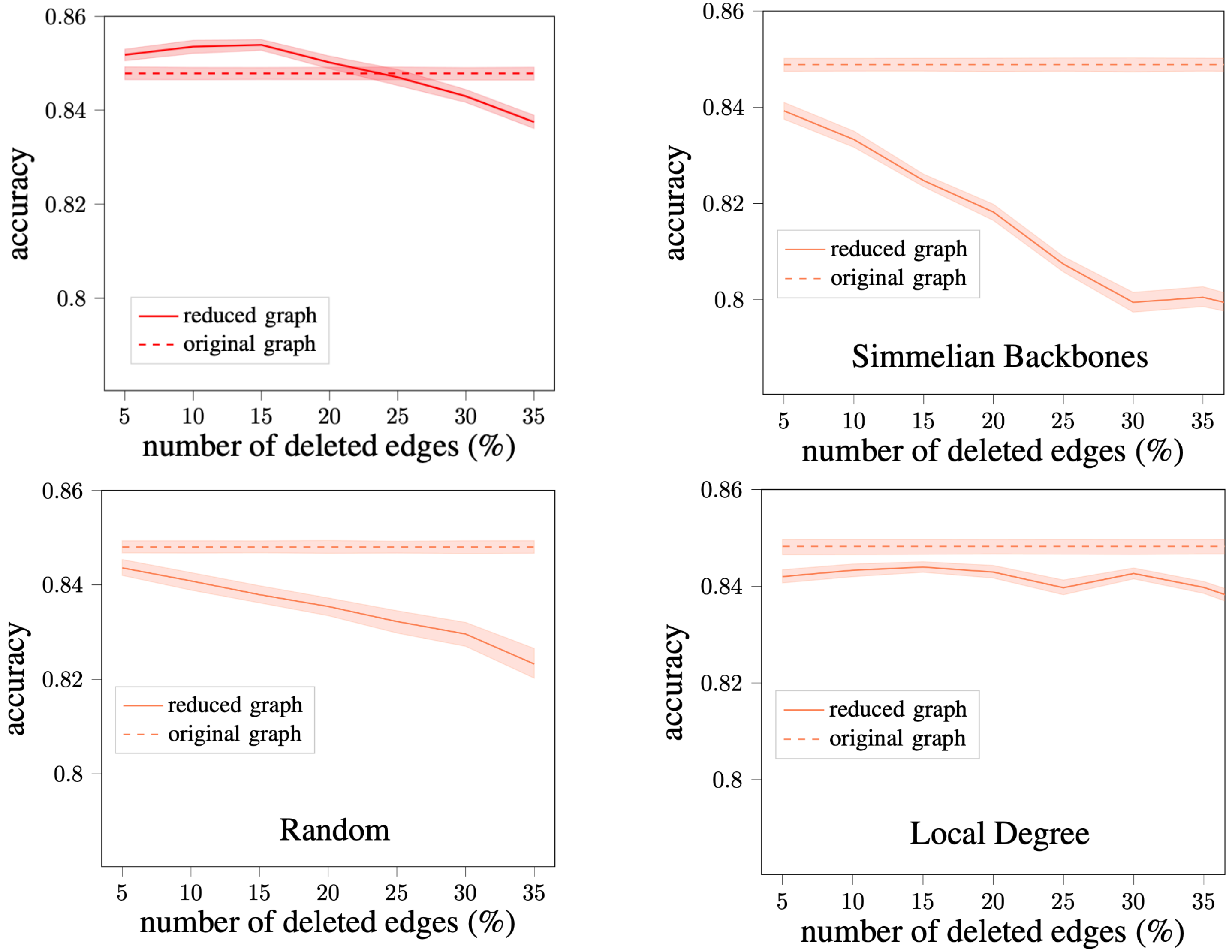
L_{train}

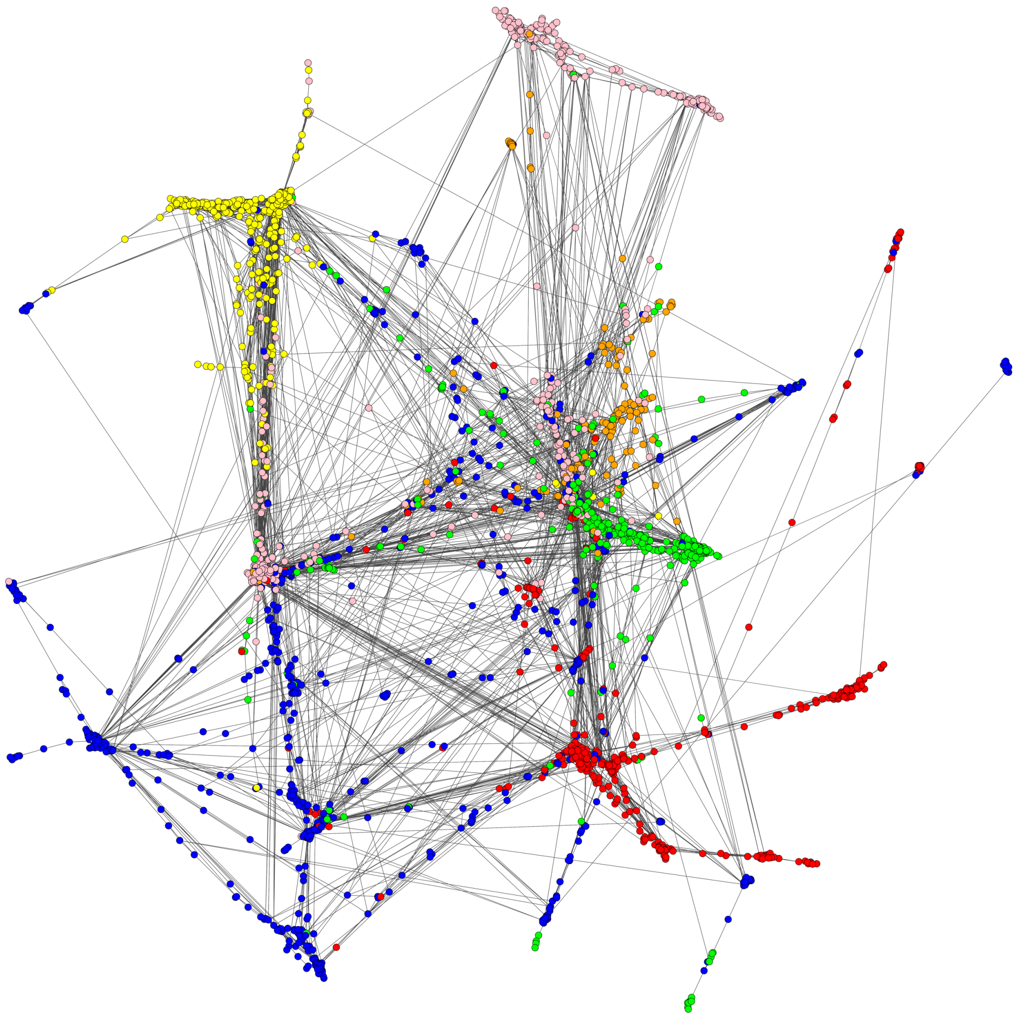
L_{self}



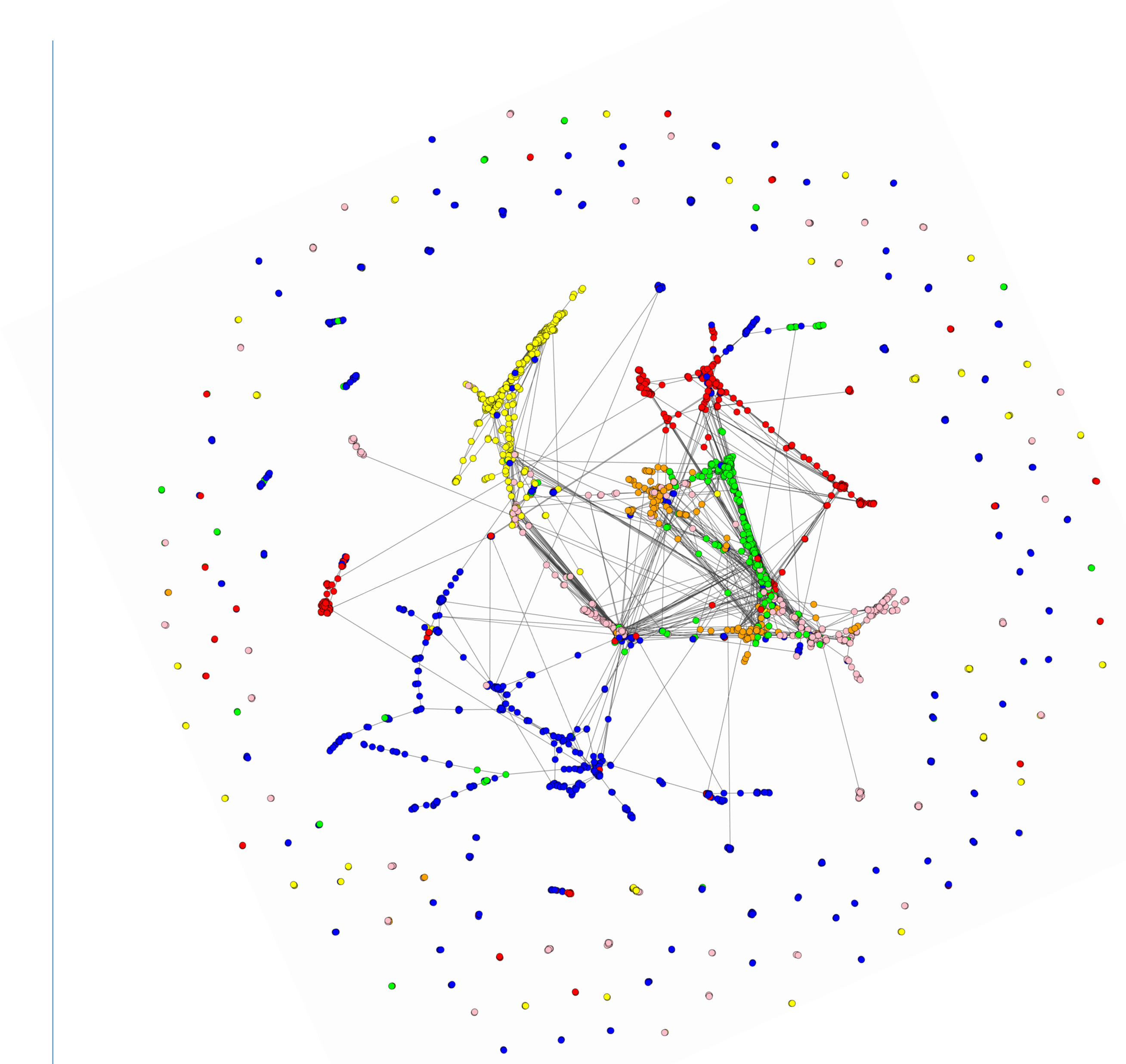
Comparison

Ours

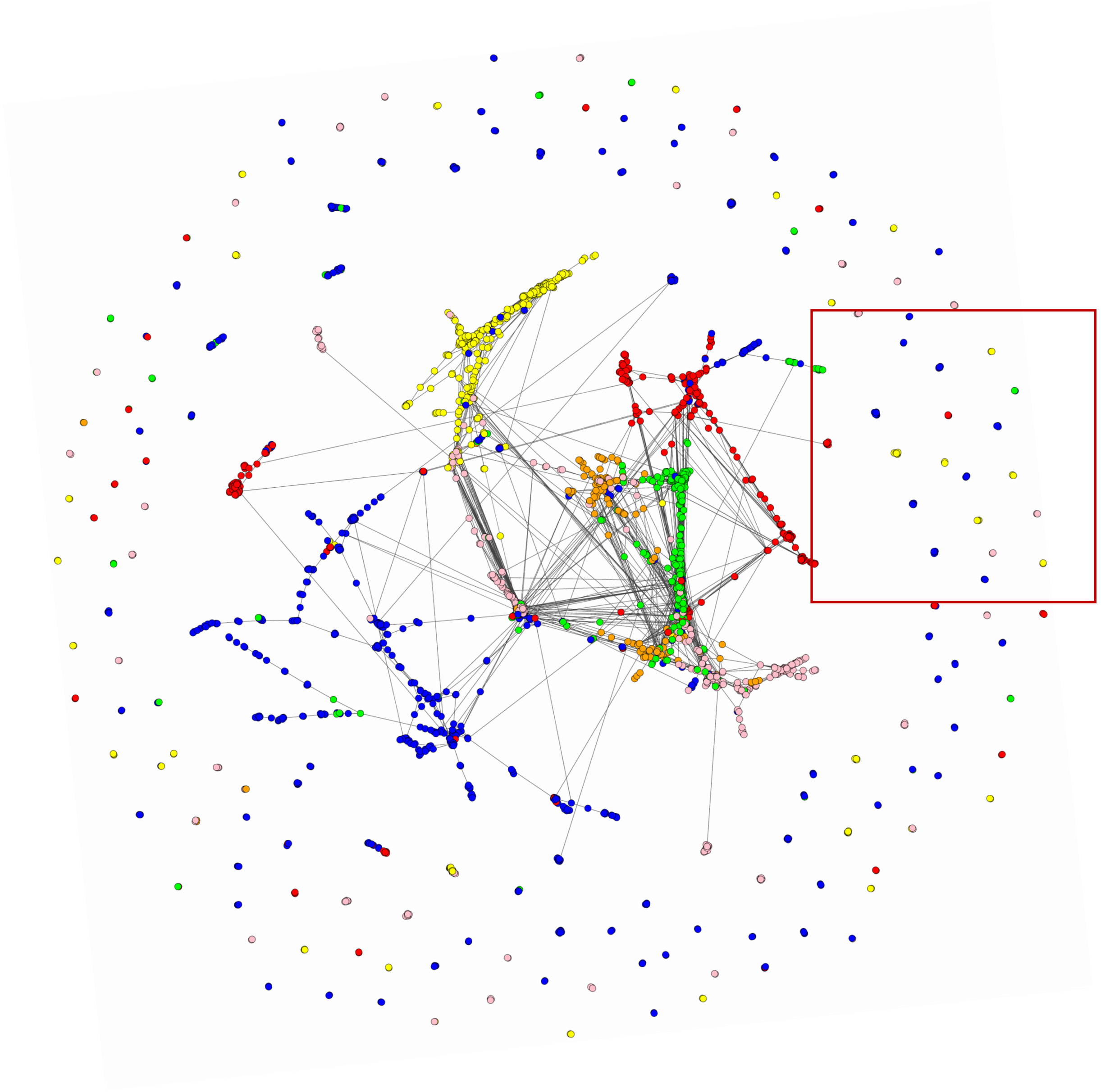


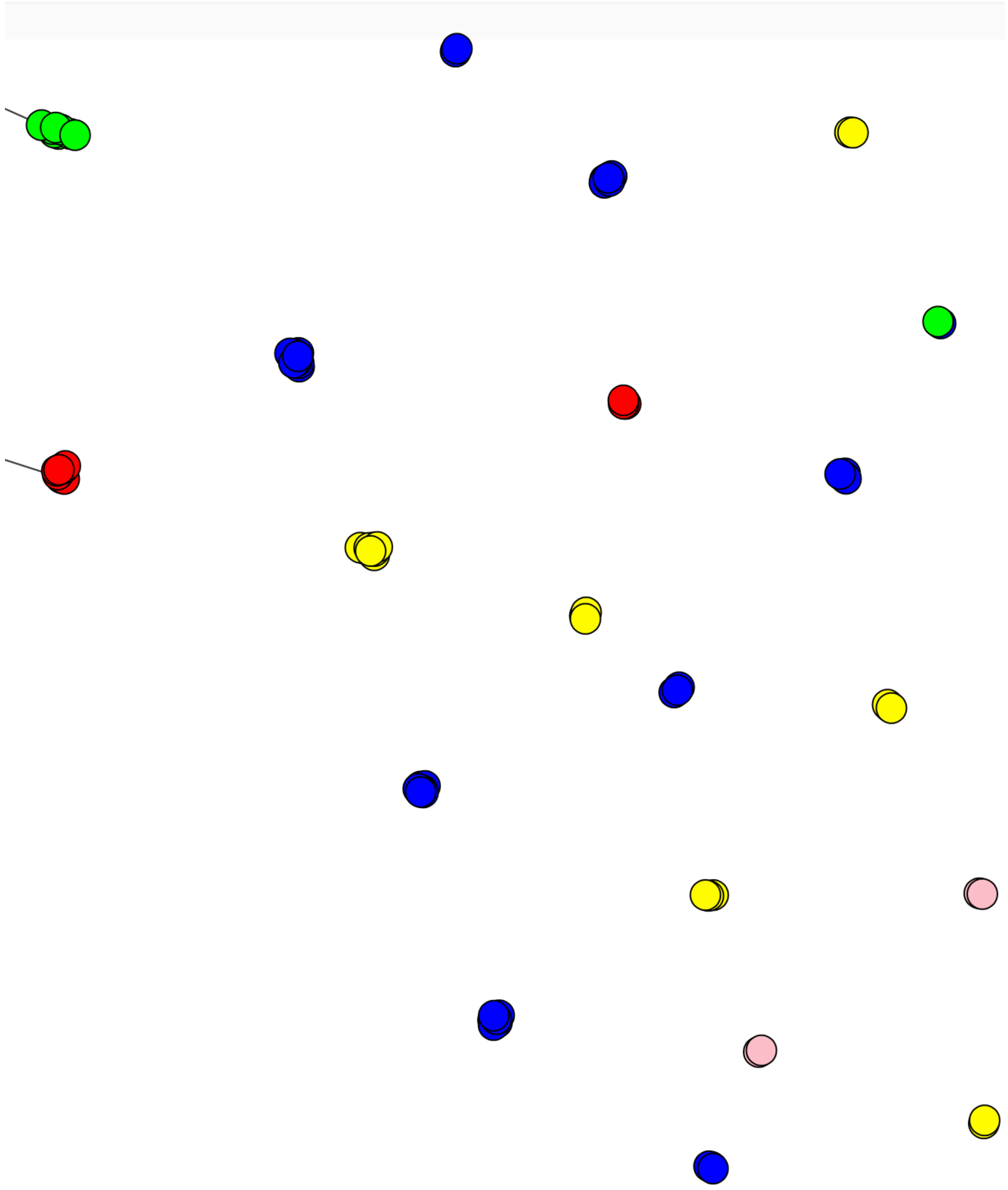


Original Graph
Test Acc : 0.848



30% edges deleted
Test Acc : 0.843





Thank You!