Group meeting – February 25, 2022

Truong Son Hy

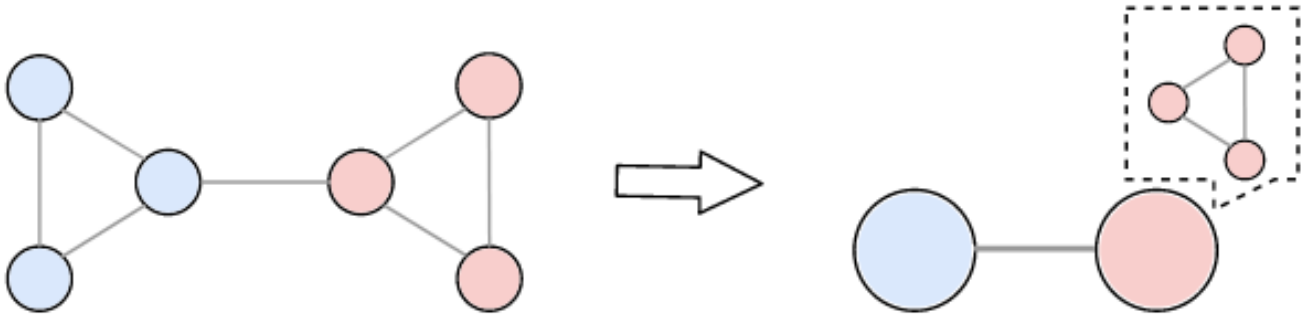
Department of Computer Science
The University of Chicago

Content

Paper:

- Chen Cai, Dingkang Wang, Yusu Wang, **Graph Coarsening with Neural Networks** (ICLR 2021).

Objective (1)



A vertex map $\pi : V \to \hat{V}$

Proposal

GNN based framework that can be trained using a collection of existing graphs in an **unsupervised** manner, so as to construct such a coarse graph $\hat{G}$ for a future input graph G (presumably coming from the same family as training graphs) that can **preserve properties** of G effectively.

Objective (2)

Note from the paper:

- As the graph G and the coarse graph $\hat{G}$ have the **different** number of nodes, their Laplace operators $\mathcal{O}_G$ and $\mathcal{O}_{\hat{G}}$ of two graphs are **not** directly comparable.
  I disagree: There is a way to compare it!

Objective (2)

Note from the paper:

- As the graph G and the coarse graph $\hat{G}$ have the **different** number of nodes, their Laplace operators $\mathcal{O}_G$ and $\mathcal{O}_{\hat{G}}$ of two graphs are **not** directly comparable.
  I disagree: There is a way to compare it!
- Instead, we will compare $\mathcal{F}(\mathcal{O}_G, f)$ and $\mathcal{F}(\mathcal{O}_{\hat{G}}, \hat{f})$, where $\mathcal{F}$ is a functional intrinsic to the graph at hand (invariant to the permutation of vertices), such as the quadratic form or Rayleigh quotient.
  Theoretically saying, it turns out that the preserved property is only the dominant eigenvalue!

Construction of the coarse graph (1)

Combinatorial Laplacian:

$$\mathcal{L} = D^{-1/2} L D^{-1/2} = I - D^{-1/2} W D^{-1/2}$$

I would say this one is the generalized version of the assignment matrix, $P \in \mathbb{R}^{n \times N}$:

$$P[r,i] = \begin{cases} \frac{1}{|\pi^{-1}(\hat{v}_r)|} & \text{if } v_i \in \pi^{-1}(\hat{v}_r) \\ 0 & \text{otherwise} \end{cases}$$

Moore-Penrose pseudoinverse $P^+ \in \mathbb{R}^{N \times n}$: $P^+[i,r] = 1$ if and only if $\pi(v_i) = \hat{v}_r$.

Construction of the coarse graph (2)

Given any linear operator A on $\mathbb{R}^N$:

$$Q_A(x) = x^T A x.$$

Proposition 3.1. *(Loukas, 2019) The combinatorial graph Laplace operator $\widehat{L} = \widehat{D} - \widehat{W}$ for the $\widehat{V}$-induced coarse graph $\widehat{G}$ constructed above equals to the operator $\tilde{L}_{\widehat{V}} = (P^+)^T L P^+$.*

Proposition 3.2. *For any vector $\hat{x} \in \mathbb{R}^n$, we have that $\mathsf{Q}_{\widehat{L}}(\hat{x}) = \mathsf{Q}_L(P^+\hat{x})$, where $\widehat{L}$ is the combinatorial Laplace operator for the $\widehat{V}$-induced coarse graph $\widehat{G}$ constructed above. That is, set $x := P^+\hat{x}$ as the lift of $\hat{x}$ in $\mathbb{R}^N$, then $\hat{x}^T \widehat{L} \hat{x} = x^T L x$.*

Construction of the coarse graph (3)

Notation of this paper becomes confusing:

- Lifting map $\mathcal{U} : \mathbb{R}^n \to \mathbb{R}^N$.
- Projection map $\mathcal{P} : \mathbb{R}^N \to \mathbb{R}^n$, where $\mathcal{P} \cdot \mathcal{U} = I$.
- $\mathcal{U} = P^+$, $\mathcal{P} = P$ and $\hat{\mathcal{O}}_{\hat{G}} = \hat{L}$.
- The size of the cluster is denoted by $\gamma_r = |\pi^{-1}(\hat{v}_r)|$.
- Let Γ be the vertex matrix, which is the $n \times n$ diagonal matrix with $\Gamma[r][r] = \gamma_{\hat{v}_r}$.
- Doublely-weighted Laplacian:

$$\widehat{\mathrm{L}} = \Gamma^{-1/2}(\widehat{D} - \widehat{W})\Gamma^{-1/2} = \Gamma^{-1/2}\widehat{L}\Gamma^{-1/2} = (P^+\Gamma^{-1/2})^T L (P^+\Gamma^{-1/2}).$$

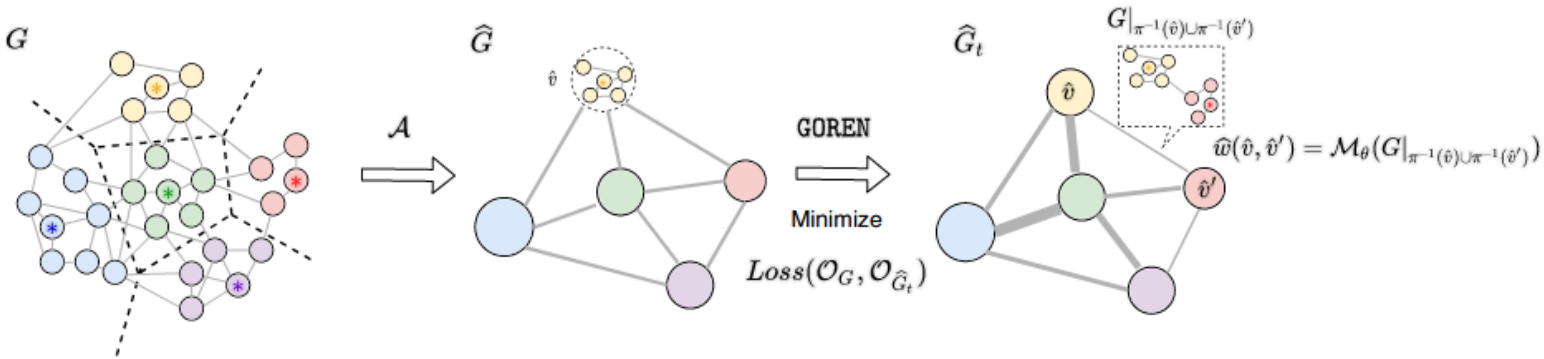
Construction of the coarse graph (4)

Proposition 3.3. *For any vector $x \in \mathbb{R}^n$, we have that $\mathsf{R}_{\widehat{\mathsf{L}}}(\hat{x}) = \mathsf{R}_L(P^+\Gamma^{-1/2}\hat{x})$. That is, set the lift of $\hat{x}$ in $\mathbb{R}^N$ to be $x = P^+\Gamma^{-1/2}\hat{x}$, then we have that $\frac{\hat{x}^T\widehat{\mathsf{L}}\hat{x}}{\hat{x}^T\hat{x}} = \frac{x^TLx}{x^Tx}$.*

Table 1: Depending on the choice of $\mathcal{F}$ (quantity that we want to preserve) and $\mathcal{O}_G$, we have different projection/lift operators and resulting $\mathcal{O}_{\widehat{G}}$ on the coarse graph.

Quantity $\mathcal{F}$ of interest	$\mathcal{O}_G$	Projection $\mathcal{P}$	Lift $\mathcal{U}$	$\mathcal{O}_{\widehat{G}}$	Invariant under $\mathcal{U}$
Quadratic form Q	L	P	P^+	Combinatorial Laplace $\widehat{L}$	$\mathsf{Q}_L(\mathcal{U}\hat{x}) = \mathsf{Q}_{\widehat{L}}(\hat{x})$
Rayleigh quotient R	L	$\Gamma^{-1/2}(P^+)^T$	$P^+\Gamma^{-1/2}$	Doubly-weighted Laplace $\widehat{\mathsf{L}}$	$\mathsf{R}_L(\mathcal{U}\hat{x}) = \mathsf{R}_{\widehat{\mathsf{L}}}(\hat{x})$
Quadratic form Q	$\mathcal{L}$	$\widehat{D}^{1/2}PD^{-1/2}$	$D^{1/2}(P^+)\widehat{D}^{-1/2}$	Normalized Laplace $\widehat{\mathcal{L}}$	$\mathsf{Q}_{\mathcal{L}}(\mathcal{U}\hat{x}) = \mathsf{Q}_{\widehat{\mathcal{L}}}(\hat{x})$

GNN-based framework (1)



Graph Isomorphism Network (GIN) to represent the learnable neural network $\mathcal{M}_\theta$ to learn a weight-assignment function μ. The model is named as Graph coarsening refinement network (GOREN).

GNN-based framework (2)

Model Architecture. The building block of our graph neural networks is based on the modification of Graph Isomorphism Network (GIN) that can handle both node and edge features. In particular, we first linear transform both node feature and edge feature to be vectors of the same dimension. At the k-th layer, GNNs update node representations by

$$h_v^{(k)} = \text{ReLU}\left(\text{MLP}^{(k)}\left(\sum_{u \in \mathcal{N}(v) \cup \{v\}} h_u^{(k-1)} + \sum_{e=(v,u): u \in \mathcal{N}(v) \cup \{v\}} h_e^{(k-1)}\right)\right) \tag{3}$$

where $\mathcal{N}(v)$ is a set of nodes adjacent to v, and $e = (v; v)$ represents the self-loop edge. Edge features $h_e^{(k-1)}$ is the same across the layers.

We use average graph pooling to obtained the graph representation from node embeddings, i.e., $h_G = \text{MEAN}\left(\left\{h_v^{(K)} | v \in G\right\}\right)$. The final prediction of weight is $1 + \text{ReLu}(\Phi(h_G))$ where Φ is a linear layer. We set the number of layers to be 3 and the embedding dimension to be 50.

GNN-based framework (3)

Given a graph G and a coarsening algorithm $\mathcal{A}$, the general form of loss is

$$Loss(\mathcal{O}_G, \mathcal{O}_{\widehat{G_t}}) = \frac{1}{k}\sum_{i=1}^{k} |\mathcal{F}(\mathcal{O}_G, f_i) - \mathcal{F}(\mathcal{O}_{\widehat{G_t}}, \mathcal{P}f_i)|,$$

where f_i is signal on the original graph (such as eigenvectors) and $\mathcal{P}f_i$ is its projection.

Laplacian of G and $\widehat{G_t}$, and the resulting *quadratic loss* has the form:

$$Loss(L, \widehat{L}_t) = \frac{1}{k}\sum_{i=1}^{k} |f_i^T L f_i - (Pf_i)^T \widehat{L}_t (Pf_i)|.$$

Fundamentally, it tries to match the dominant eigenvalue.

GNN-based framework (3)

Given a graph G and a coarsening algorithm $\mathcal{A}$, the general form of loss is

$$Loss(\mathcal{O}_G, \mathcal{O}_{\widehat{G_t}}) = \frac{1}{k}\sum_{i=1}^{k} |\mathcal{F}(\mathcal{O}_G, f_i) - \mathcal{F}(\mathcal{O}_{\widehat{G_t}}, \mathcal{P}f_i)|,$$

where f_i is signal on the original graph (such as eigenvectors) and $\mathcal{P}f_i$ is its projection.

Laplacian of G and $\widehat{G_t}$, and the resulting *quadratic loss* has the form:

$$Loss(L, \widehat{L}_t) = \frac{1}{k}\sum_{i=1}^{k} |f_i^T L f_i - (Pf_i)^T \widehat{L}_t (Pf_i)|.$$

Fundamentally, it tries to match the dominant eigenvalue.

Finally, given a collection of training graphs $G_1, \ldots, G_m$, we will train for parameters in the module $\mathcal{M}_\theta$ to minimize the total loss on training graphs. When a test graph G_{test} is given, we simply apply $\mathcal{M}_\theta$ to set up weight for each edge in $\widehat{G_{test}}$, obtaining a new graph $\widehat{G_{test,t}}$. We compare $Loss(\mathcal{O}_{G_{test}}, \mathcal{O}_{\widehat{G_{test,t}}})$ against $Loss(\mathcal{O}_{G_{test}}, \mathcal{O}_{\widehat{G_{test}}})$ and expect the former loss is smaller.

GNN-based framework (4)

Recall our goal is to

$$\begin{aligned} \underset{\mathbf{w},U}{\text{minimize}} \quad & \left\| \mathfrak{L}\mathbf{w} - U \operatorname{Diag}(\boldsymbol{\lambda}) U^T \right\|_F^2 \\ \text{subject to} \quad & \mathbf{w} \geq 0, U^T U = I \end{aligned} \tag{5}$$

Algorithm 1: Iterative algorithm for edge weight optimization

Input: coarse graph $\widehat{G}$, error tolerance ϵ, iteration limit T
Output: coarse graph with new edge weights

1 Initialize U as random element in orthogonal group $O(n, \mathbb{R})$ and $t = 0$.
2 **while** ϵ *is smaller than the threshold or* $t > T$ **do**
3 Update $\mathbf{w}^{t+1}, U^{t+1}$ according to 8 and Lemma F.4
4 Compute Error ϵ
5 $t = t + 1$
6 From w^t, output coarse graph with new edge weights.

where $\boldsymbol{\lambda}$ is the desired eigenvalues of the smaller graph. One choice of $\boldsymbol{\lambda}$ can be the first n eigenvalues of the original graph of size N. $\mathbf{w}$ and U are variables of size $n(n-1)/2$ and $n \times n$.

The algorithm proceeds by iteratively updating U and $\mathbf{w}$ while fixing the other one.

If we can improve the graph coarsening algorithm by an iterative method (trying to match the desired spectrum) then why do we need GNN in the first place?

Experiments (1)

Heavy Edge Matching. At each level of the scheme, the contraction family is obtained by computing a maximum-weight matching with the weight of each contraction set (v_i, v_j) calculated as $w_{ij}/\max\{d_i, d_j\}$. In this manner, heavier edges connecting vertices that are well separated from the rest of the graph are contracted first.

Algebraic Distance. This method differs from heavy edge matching in that the weight of each candidate set $(v_i, v_j) \in E$ is calculated as $\left(\sum_{q=1}^{Q} \left(x_q(i) - x_q(j)\right)^2\right)^{1/2}$, where x_k is an N-dimensional test vector computed by successive sweeps of Jacobi relaxation. The complete method is described by Ron et al. (2011), see also Chen & Safro (2011).

Affinity. This is a vertex proximity heuristic in the spirit of the algebraic distance that was proposed by Livne & Brandt (2012) in the context of their work on the lean algebraic multigrid. As per the author suggests, the $Q = k$ test vectors are here computed by a single sweep of a Gauss-Seidel iteration.

Local Variation. There are two variations of local variation methods, edge-based local variation, and neighborhood-based local variation. They differ in how the contraction set is chosen. Edge-based variation is constructed for each edge, while the neighborhood-based variant takes every vertex and its neighbors as contraction set. What two methods have common is that they both optimize an upper bound of the restricted spectral approximation objective. In each step, they greedily pick the sets whose local variation is the smallest. See Loukas (2019) for more details.

Baseline. We also implement a simple baseline that randomly chooses a collection of nodes in the original graph as landmarks and contract other nodes to the nearest landmarks. If there are multiple nearest landmarks, we randomly break the tie. The weight of the coarse graph is set to be the sum of the weights of the crossing edges.

Competing methods

Experiments (2)

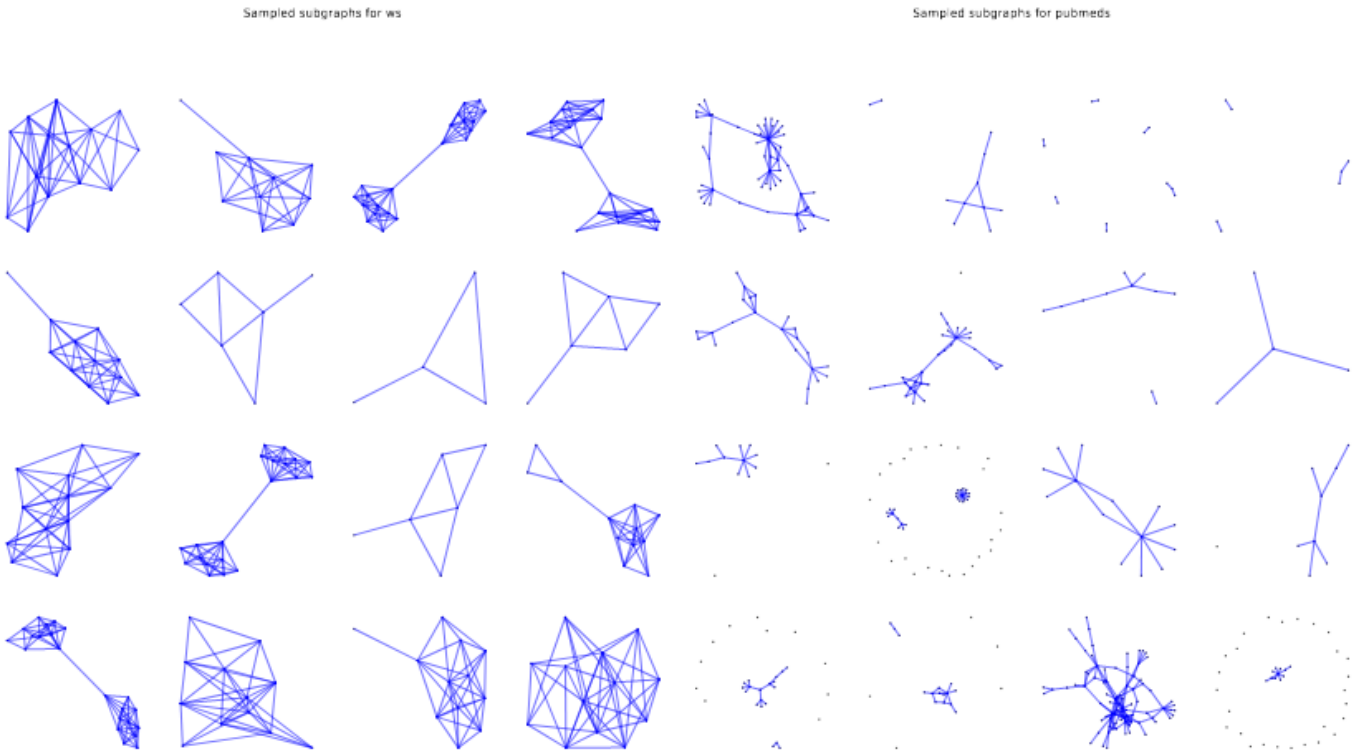


Figure 4: A collection of subgraphs corresponding to edges in coarse graphs (WS and PubMed) generated by variation neighborhood algorithm. Reduction ratio is 0.7 and 0.9 respectively.

Experiments (3)

The authors try to improve the existing state-of-the-art graph coarsening algorithm $\mathcal{A}$ from (Loukas, 2019):

Table 2: The error reduction after applying GOREN.

Dataset	Affinity	Algebraic Distance	Heavy Edge	Local var (edges)	Local var (neigh.)
Airfoil	91.7%	88.2%	86.1%	43.2%	73.6%
Minnesota	49.8%	57.2%	30.1%	5.50%	1.60%
Yeast	49.7%	51.3%	37.4%	27.9%	21.1%
Bunny	84.7%	69.1%	61.2%	19.3%	81.6%

Experiments (4)

Table 5: Loss: Eigenerror. Laplacian: combinatorial Laplacian for original graphs and doubly-weighted Laplacian for coarse ones. Each entry $x(y)$ is: x = loss w/o learning, and y = improvement percentage. † stands for out of memory.

	Dataset	BL	Affinity	Algebraic Distance	Heavy Edge	Local var (edges)	Local var (neigh.)
Synthetic	BA	0.36 (7.1%)	0.17 (8.2%)	0.22 (6.5%)	0.22 (4.7%)	0.11 (21.1%)	0.17 (-15.9%)
	ER	0.61 (0.5%)	0.70 (1.0%)	0.35 (0.6%)	0.36 (0.2%)	0.19 (1.2%)	0.02 (0.8%)
	GEO	1.72 (50.3%)	0.16 (89.4%)	0.18 (91.2%)	0.45 (84.9%)	0.08 (55.6%)	0.20 (86.8%)
	WS	1.59 (43.9%)	0.11 (88.2%)	0.11 (83.9%)	0.58 (23.5%)	0.10 (88.2%)	0.12 (79.7%)
Real	CS	1.10 (18.0%)	0.55 (49.8%)	0.33 (60.6%)	0.42 (44.5%)	0.21 (75.2%)	0.0 (-154.2%)
	Flickr	0.57 (55.7%)	†	0.33 (20.2%)	0.31 (55.0%)	0.11 (67.6%)	0.07 (60.3%)
	Physics	1.06 (21.7%)	0.58 (67.1%)	0.33 (69.5%)	0.35 (64.6%)	0.20 (79.0%)	0.0 (-377.9%)
	PubMed	1.25 (7.1%)	0.50 (15.5%)	0.51 (12.3%)	1.19 (-110.1%)	0.35 (-8.8%)	0.02 (60.4%)
	Shape	2.07 (67.7%)	0.24 (93.3%)	0.17 (90.9%)	0.49 (93.0%)	0.11 (84.2%)	0.20 (90.7%)

I think this is the most relevant (theoretically supported) result!