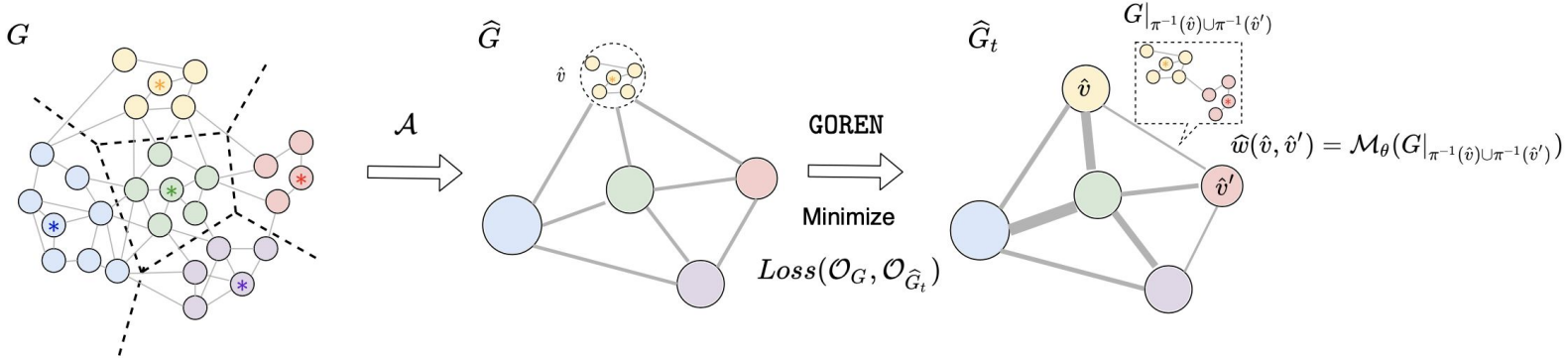
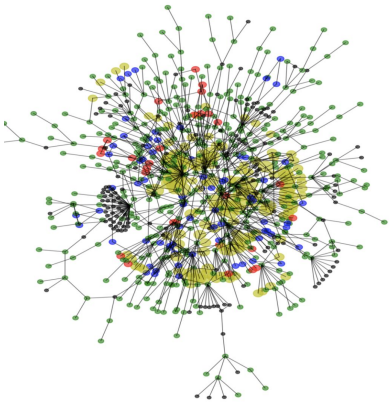
Graph Coarsening with Neural Networks

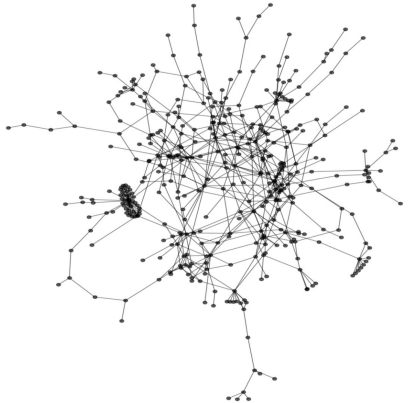
Chen Cai*, Dingkang Wang, Yusu Wang



Motivation

- Make a small graph out of a large graph while preserving some properties
- Fundamental operation
- Useful for visualization, scientific computation, and other downstream tasks
- Teng & Spielman's famous edge sparsification algorithm





Graph coarsening

- You can not preserve everything in general. So what properties are you considering?
- Spectral property!

$$G \underset{\mathcal{U}}{\overset{\mathcal{P}}{\rightleftarrows}} \hat{G}$$

- Define projection/lift operator, graphs operator, and their properties

How to measure the quality?

- Compare $\mathcal{F}(\mathcal{O}_G, f)$ and $\mathcal{F}(\mathcal{O}_{\widehat{G}}, \hat{f})$
- $\mathcal{F}$ can be quadratic form or Rayleigh quotient

$$x^T L x \qquad \frac{x^T L x}{x^T x}$$

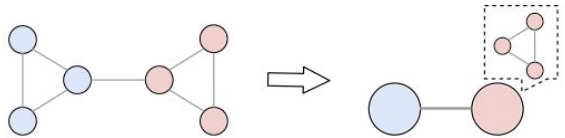
- $O_G, O_{\widehat{G}}$ are the Laplace operators
- f is graph signal such as the eigenvalues of graph Laplacian

Example

$$\mathcal{P} = P = \begin{bmatrix} 1/3 & 1/3 & 1/3 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1/3 & 1/3 & 1/3 \end{bmatrix} \qquad \mathcal{U} = P^{+} = \begin{bmatrix} 1 & 0 \\ 1 & 0 \\ 1 & 0 \\ 0 & 1 \\ 0 & 1 \\ 0 & 1 \end{bmatrix}$$

Proposition A.2. *For any vector $\hat{x} \in \mathbb{R}^n$, we have that $\mathsf{Q}_{\widehat{L}}(\hat{x}) = \mathsf{Q}_L(P^{+}\hat{x})$. In other words, set $x := P^{+}\hat{x}$ as the lift of $\hat{x}$ in $\mathbb{R}^N$, then $\hat{x}^T\widehat{L}\hat{x} = x^T L x$.*

Proof. $\mathsf{Q}_L(\mathcal{U}\hat{x}) = (\mathcal{U}\hat{x})^T L \mathcal{U}\hat{x} = \hat{x}(P^{+})^T L P^{+}\hat{x}^T = \hat{x}^T\widehat{L}\hat{x} = \mathsf{Q}_{\widehat{L}}(\hat{x})$ □

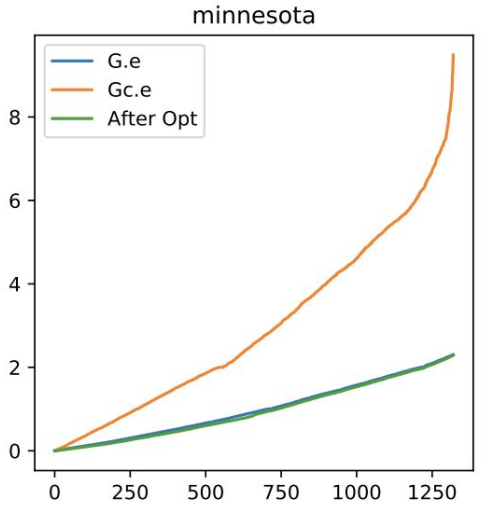


Invariant

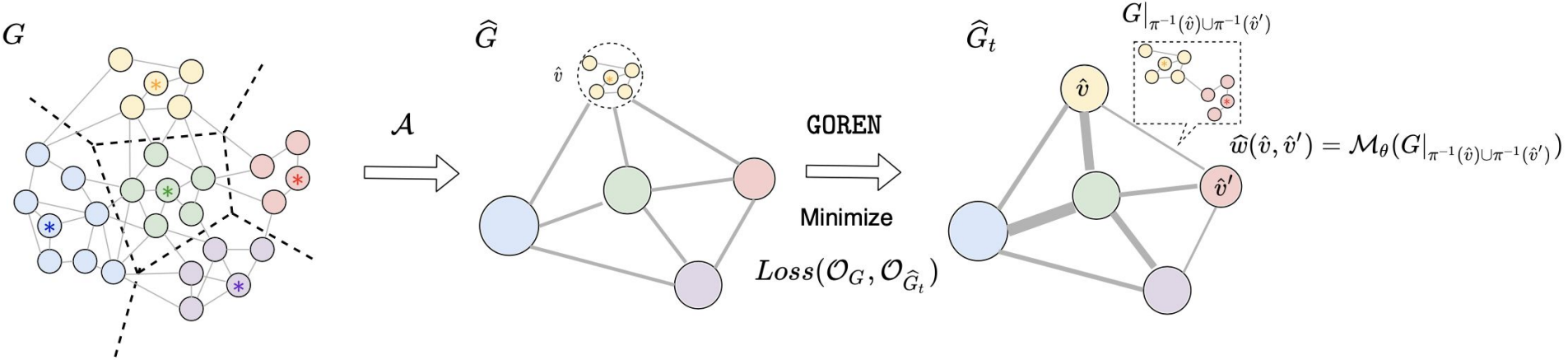
Quantity $\mathcal{F}$ of interest	$\mathcal{O}_G$	Projection $\mathcal{P}$	Lift $\mathcal{U}$	$\mathcal{O}_{\widehat{G}}$	Invariant under $\mathcal{U}$
Quadratic form Q	L	P	P^+	Combinatorial Laplace $\widehat{L}$	$\mathsf{Q}_L(\mathcal{U}\hat{x}) = \mathsf{Q}_{\widehat{L}}(\hat{x})$
Rayleigh quotient 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 $\widehat{\mathcal{L}}$	$\mathsf{Q}_{\mathcal{L}}(\mathcal{U}\hat{x}) = \mathsf{Q}_{\widehat{\mathcal{L}}}(\hat{x})$

Key Observation

- Existing coarsening algorithm does not optimize for edge weight
- Theory: convergence result
- Practice: nearly identical eigenvalues alignment after optimization
- So let’s learn the edge weight
  - cvx. slow and does not generalize
  - Neural network: suboptimal but generalize



Graph cOarsening RefinemEnt Network (GOREN)



Experiments

- Extensive experiments on synthetic graphs and real networks
- Synthetic graphs from common generative models
- Real networks: shape meshes; citation networks; largest one has 89k nodes

Table 3: Loss: quadratic loss. Laplacian: combinatorial Laplacian for both original and coarse graphs. Each entry $x(y)$ is: $x =$ loss w/o learning, and $y =$ improvement percentage.

	Dataset	BL	Affinity	Algebraic Distance	Heavy Edge	Local var (edges)	Local var (neigh.)
Synthetic	BA	0.44 (16.1%)	0.44 (4.4%)	0.68 (4.3%)	0.61 (3.6%)	0.21 (14.1%)	0.18 (72.7%)
	ER	0.36 (1.1%)	0.52 (0.8%)	0.35 (0.4%)	0.36 (0.2%)	0.18 (1.2%)	0.02 (7.4%)
	GEO	0.71 (87.3%)	0.20 (57.8%)	0.24 (31.4%)	0.55 (80.4%)	0.10 (59.6%)	0.27 (65.0%)
	WS	0.45 (62.9%)	0.09 (82.1%)	0.09 (60.6%)	0.52 (51.8%)	0.09 (69.9%)	0.11 (84.2%)
Real	CS	0.39 (40.0%)	0.21 (29.8%)	0.17 (26.4%)	0.14 (20.9%)	0.06 (36.9%)	0.0 (59.0%)
	Flickr	0.25 (10.2%)	0.25 (5.0%)	0.19 (6.4%)	0.26 (5.6%)	0.11 (11.2%)	0.07 (21.8%)
	Physics	0.40 (47.4%)	0.37 (42.4%)	0.32 (49.7%)	0.14 (28.0%)	0.15 (60.3%)	0.0 (-0.3%)
	PubMed	0.30 (23.4%)	0.13 (10.5%)	0.12 (15.9%)	0.24 (10.8%)	0.06 (11.8%)	0.01 (36.4%)
	Shape	0.23 (91.4%)	0.08 (89.8%)	0.06 (82.2%)	0.17 (88.2%)	0.04 (80.2%)	0.08 (79.4%)

Experiment

- Generalize to graph from same generative model
- Train on small subgraph, generalize to much large (**25x**) graphs
- Works for different objective functions, both differentiable or non-differentiable

Table 4: Loss: quadratic loss. Laplacian: normalized Laplacian for original and coarse graphs. Each entry $x(y)$ is: $x =$ loss w/o learning, and $y =$ improvement percentage.

	Dataset	BL	Affinity	Algebraic Distance	Heavy Edge	Local var (edges)	Local var (neigh.)
Synthetic	BA	0.13 (76.2%)	0.14 (45.0%)	0.15 (51.8%)	0.15 (46.6%)	0.14 (55.3%)	0.06 (57.2%)
	ER	0.10 (82.2%)	0.10 (83.9%)	0.09 (79.3%)	0.09 (78.8%)	0.06 (64.6%)	0.06 (75.4%)
	GEO	0.04 (52.8%)	0.01 (12.4%)	0.01 (27.0%)	0.03 (56.3%)	0.01 (-145.1%)	0.02 (-9.7%)
	WS	0.05 (83.3%)	0.01 (-1.7%)	0.01 (38.6%)	0.05 (50.3%)	0.01 (40.9%)	0.01 (10.8%)
Real	CS	0.08 (58.0%)	0.06 (37.2%)	0.04 (12.8%)	0.05 (41.5%)	0.02 (16.8%)	0.01 (50.4%)
	Flickr	0.08 (-31.9%)	0.06 (-27.6%)	0.06 (-67.2%)	0.07 (-73.8%)	0.02 (-440.1%)	0.02 (-43.9%)
	Physics	0.07 (47.9%)	0.06 (40.1%)	0.04 (17.4%)	0.04 (61.4%)	0.02 (-23.3%)	0.01 (35.6%)
	PubMed	0.05 (47.8%)	0.05 (35.0%)	0.05 (41.1%)	0.12 (46.8%)	0.03 (-66.4%)	0.01 (-118.0%)
	Shape	0.02 (84.4%)	0.01 (67.7%)	0.01 (58.4%)	0.02 (87.4%)	0.0 (13.3%)	0.01 (43.8%)

Future work

- Combine with downstream tasks
  - Graph classification; scientific applications; combinatorial optimization on graphs…
- Optimize for node grouping operation
  - Reinforcement learning?
  - Gumbel-Softmax
- Extend to complex networks with node/edge features

Thank you!