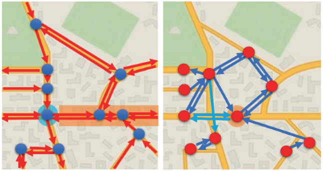
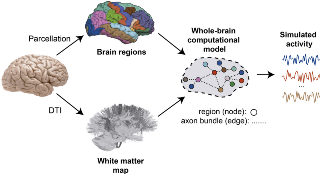
Efficient and scalable graph generation through iterative local expansion

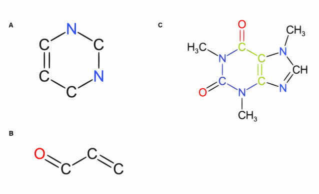
Luka Lafaye de Micheaux

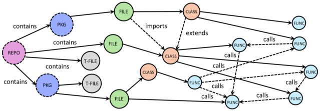
August 21, 2025

Introduction







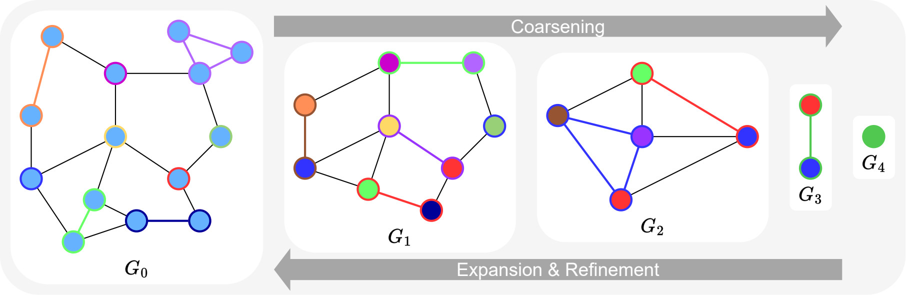


Source: Bachechi et al. 2022 (Road Network Graph), Susi et al. 2021 (FNS Neuron model), Wikipedia Commons, Tao et al. 2025 (CGM model)

Problem statement

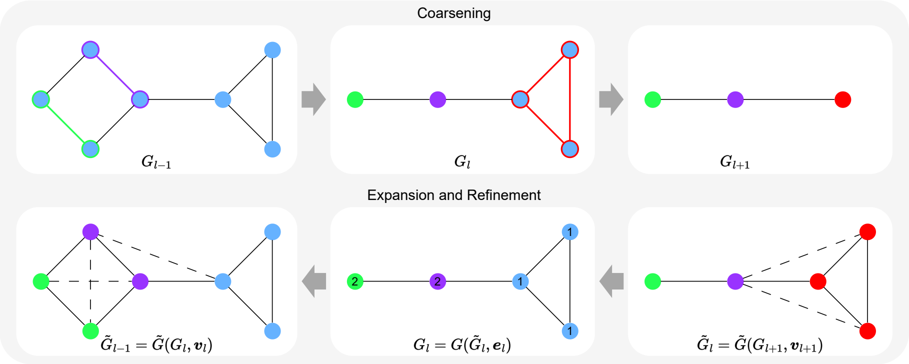
- Given real-world graphs sampled from an unknown distribution $\mathcal{G}_{\text{real}}$, we want to learn a generative model $p_\theta(G)$ that approximates this distribution.
- The sampled new graphs $G \sim p_\theta$ from the model should be similar to those from $\mathcal{G}_{\text{real}}$.
- Challenges:
  - Modeling the full joint distribution over all node pairs,
  - Capturing both local patterns and global structure,
  - Scaling to large graphs and generalizing to unseen sizes.

Iterative coarsening, expansion, and refinement



Source: Bergmeister et al. 2024 - llustration of the iterative coarsening, expansion, and refinement process.

Notations of intermediate states



Source: Bergmeister et al. 2024 - Notations of intermediate steps during the process.

Likelihood of a graph

- Coarsening provides a sequence $G^{(0)} = G,\ G^{(1)}, \dots G^{(L-1)},\ G^{(L)}$ where $G^{(L)}$ is a single node graph
- Expansion and refinement provides the inverse sequence $G^{(L)}$, $\widetilde{G}^{(L-1)},\ G^{(L-1)}, \dots, \widetilde{G}^{(1)},\ G^{(1)}, \widetilde{G}^{(0)}, G^{(0)} = G$
- Let $\omega \in E(G)$ denote an expansion-refinement sequence that reconstructs G, and $\pi \in C(G)$ a coarsening sequence ending in a singleton.
- Then the likelihood of G is defined as:

$$p(G) = \sum_{\omega \in E(G)} p(\omega) > \sum_{\pi \in C(G)} p(\pi)$$

Contraction families and elbow trick

- We restrict possible contraction sets to belong to $\mathcal{F}(G)$ (edge contractions in my implementation), and coarsening sequences are sampled in $C_{\mathcal{F}}(G)$. Hence:

$$p(G) = \sum_{\omega \in E(G)} p(\omega) > \sum_{\pi \in C(G)} p(\pi) > \sum_{\pi \in C_{\mathcal{F}}(G)} p(\pi)$$

- Given a distribution $q(\pi \mid G)$ over coarsening sequences $C_{\mathcal{F}}(G)$,

$$p(G) > \mathbb{E}_{\pi \sim q(\pi \mid G)} \left[\frac{p(\pi)}{q(\pi \mid G)} \right]$$

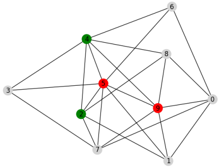
- Using the elbow trick we have:

$$\log(p(G)) > \mathbb{E}_{\pi \sim q(\pi \mid G)} \left[\log(p(\pi)) - \log(q(\pi \mid G)) \right]$$

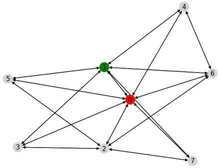
Implementation details

- Sampling $\pi \sim q(\pi \mid G)$ provides coarsening sequences from which we can extract $G^{(L)}, \widetilde{G}^{(L-1)}(v_L), G^{(L-1)}(e_{L-1}), \ldots, \widetilde{G}^{(1)}(v_2), G^{(1)}(e_1), \widetilde{G}^{(0)}(v_1), G^{(0)}(e_0) = G$
- The dataset contains pairs of (v_l, e_l) stored in $\widetilde{G}^{(l}$, resp. as node and edge features. They are noised into $\left(\hat{e}_l, \hat{v}_l\right)$ during training.
- The denoising model learns to recover (e_l, v_l) given $\widetilde{G}^{(l)}$. It consists of two GCN layers followed by two MLP heads, one used to recover v_l, and the other used to recover e_l.
- Implementation was done from scratch

Simplified edge selection cost function



Before merge



After merge

- Each eigenvector of the Laplacian corresponds to a mode
- Signals x get propagated in the graph using $D^{-1}A \times x$
- Each signal can be decomposed in modes (one per eigenvector). $e^l(i)$ corresponds to how much node i influences mode of eigenvector e^l
- Let all eigenvectors of L be noted e^i. We use:

$$\mathrm{Cost}(i,j) = \frac{1}{N-1} \sum_{l=1}^{N-1} \left(e^l(i) - e^l(j)\right)^2.$$

- Three GCN layers

$$
\begin{aligned}
h^{(1)} &= \mathrm{ReLU}\big(\mathrm{GCNConv}_1(\hat{v}_l)\big), \\
h^{(2)} &= \mathrm{ReLU}\big(\mathrm{GCNConv}_2(h^{(1)})\big), \\
h^{(3)} &= \mathrm{ReLU}\big(\mathrm{GCNConv}_3(h^{(2)})\big).
\end{aligned}
$$

- Node noise head

$$\hat{\varepsilon}_v = \mathrm{MLP}_{\mathrm{node}}\big(h^{(3)}\big)$$

- Edge noise head

$$\hat{\varepsilon}_e = \mathrm{MLP}_{\mathrm{edge}}\big([h_i^{(3)},\ h_j^{(3)},\ e_{ij}^{(t)}]\big)$$

- GCN Reminder: $H^{(l+1)} = \sigma\Big(\tilde{D}^{-\frac{1}{2}}\,\tilde{A}\,\tilde{D}^{-\frac{1}{2}}\,H^{(l)}\,W^{(l)}\Big)$ where $\tilde{A} = A + I,\ \tilde{D}_{ii} = \sum_j \tilde{A}_{ij}$.

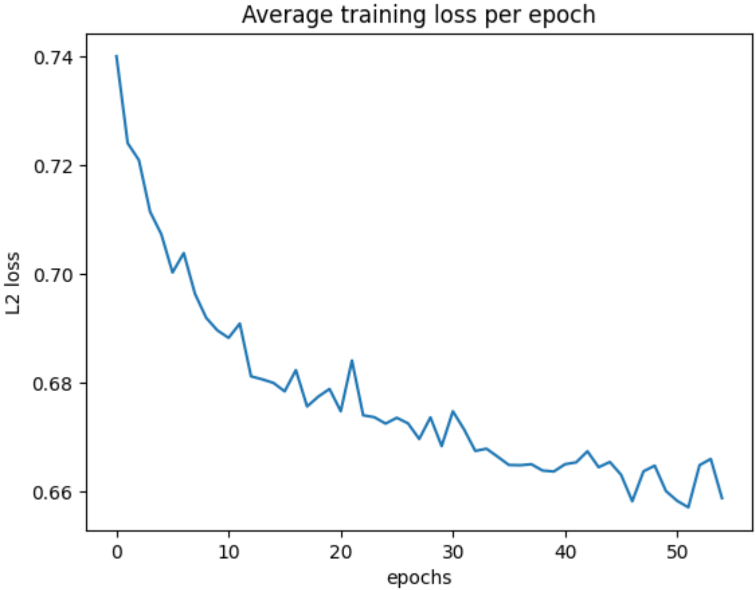
Improving model w.r.t. L2 loss on Erdos-Renyi dataset

- L2 loss on random noise sampled $X \sim \mathcal{N}(0, 1)$, $P(|X| \leq 3) = 99.7\%$.
- Initial learning rate 10e-5 and 60 epochs obtained average loss of 0.97.
- Adjusted learning rate to 10e-4 and obtained average loss 0.85
- Added sinusoidal encodings instead of linearly projecting timestep scalar and got average loss of 0.70
- Added norm layers after each GCN conv and dropout after last conv and obtained average loss of 0.65

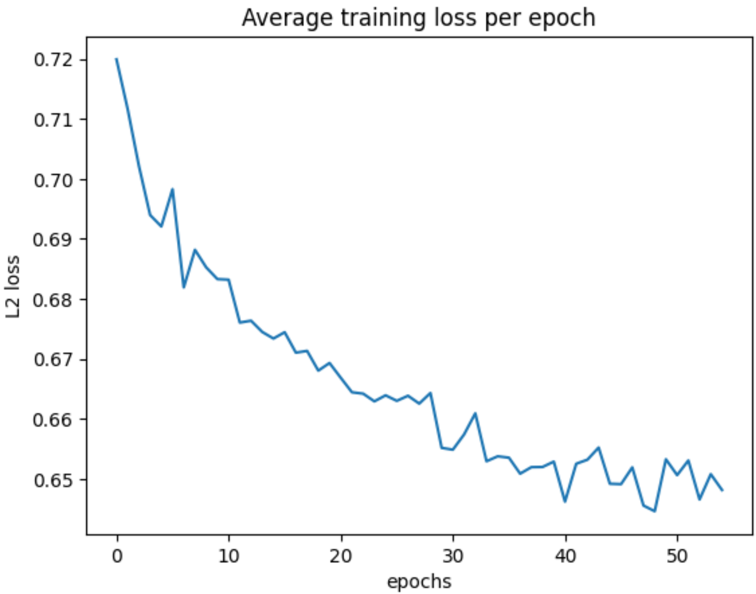
Experiments

- Erdos-Renyi synthetic dataset (n=10, p=0.5)
  - Average degree
  - Average clustering coefficients
  - Average shortest path
  - Degree distribution
- Planar graphs dataset using plantri (c=3m n=12)
  - Density
  - Planarity proportion

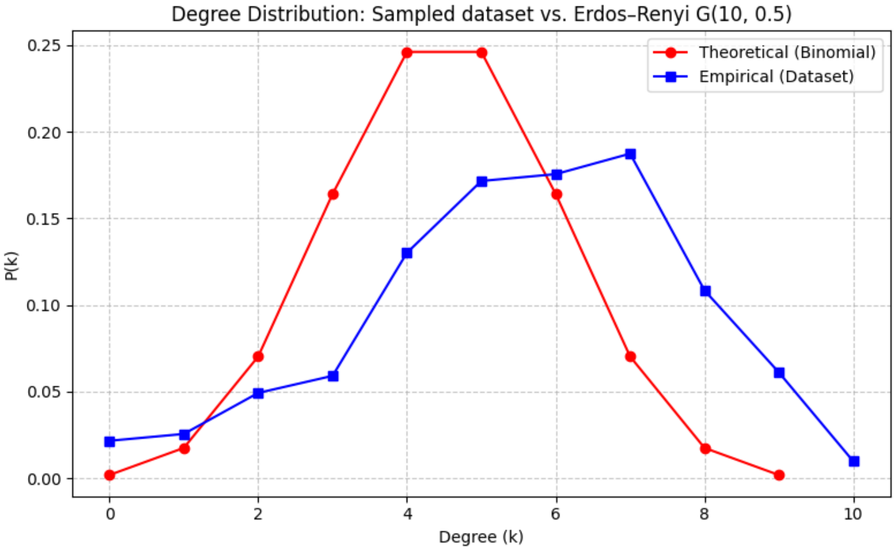
Training loss on Erdos-Renyi dataset



Training loss on planar dataset

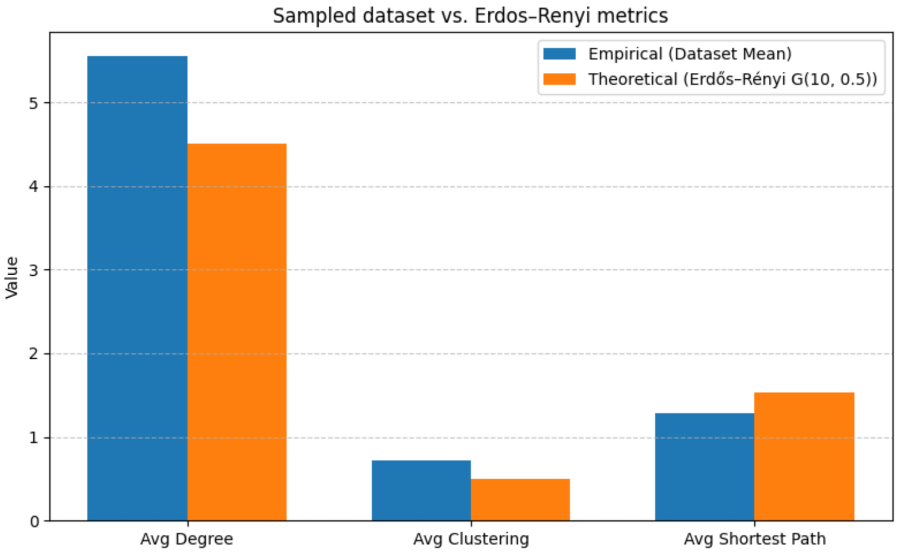


Results - Erdos Renyi



Problem with self edges

Results - Erdos Renyi



Results - Planar

- Proportion of planar graphs: 10%
- Average density: 0.410 (num edges 41% of complete graph)
- Harder task, bad results explained by lack of edge features

Bonus: improving results

- Improve edge selection cost function, use neighborhoods coarsening instead of edge coarsening
- Try Edge-GAT and leverage edge features
- Randomly skip/connect nodes during message passing
- Denoise graphs while training and add loss term that penalizes crossings?