Machine Learning with graphs - Project Defense

Delaunay Graph: Addressing Over-Squashing and Over-Smoothing Using Delaunay Triangulation
by Attali H., Duscaldi D. and Pernelle N. [2]

Edwin Roussin and Tristan Waddington

Supervised by Jhony H. Giraldo
IP-Paris, CEMST

26/03/2025

Delauney triangulation

Reconstruct a graph completely from projected features using the Delaunay triangulation.
⇒ Avoid **over-smoothing** and **over-squashing**.

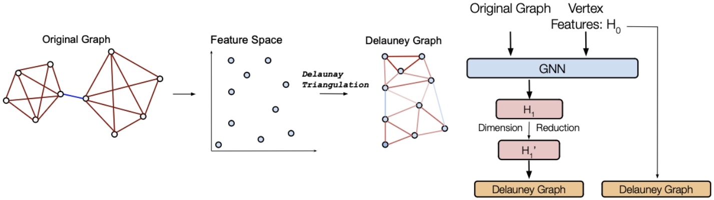


Figure: Illustration of the Delaunay rewiring [2, Attali al., 2024]

Outline

Need of Graph Rewiring

Over-Squashing: inefficient information propagation

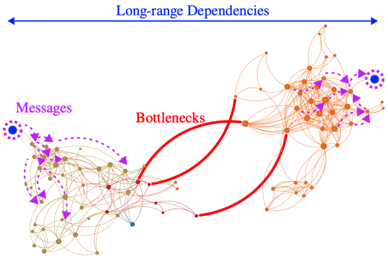


Figure: Illustration of Bottlenecks [Giraldo, Lecture GNNs, 2025]

GNNs struggle to propagate info to distant nodes: **bottleneck** when aggregating messages across a long path [1, Alon et al., 2021].
Causes **over-squashing** of exponentially growing info into fixed-size vectors. $\Rightarrow$ *Perform poorly when prediction task depends on long-range interaction.*

Vulnerable GNNs

GCNx *absorb incoming edges equally*, more susceptible to over-squashing than GAT.

Curvature metric

Negative *Discrete Ricci curvature* [8, Topping et al. 2021] to identify bottlenecks.

Over-Smoothing: consequence of message passing paradigm

Message-passing neural networks (MPNN):

Iterative approach, updating node representations through the local aggregation of information from neighboring nodes.

Causes **over-smoothing** by the need to stack additional layers to capture non-local interactions. Will smooth-out heterophilic graphs. $\Rightarrow$ *Nodes' representations are similar.*

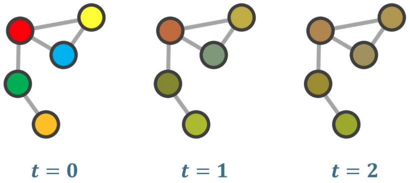


Figure: Illustration of Over-smoothing by Alex Ganose

Existing Solutions

Identify the quality of the message passing:

- **Graph structure analysis** using curvature, but does not scale.
  Highly positive curved edges $\rightarrow$ over-smoothing [5, Nguyen et al., 2023].
  Highly negative curved edges $\rightarrow$ over-squashing [8, Topping et al., 2021].
- **Need original graph** but sometimes only features available (NER, documents, ...).

Avoid over-smoothing in preventing the embedding to become the same:

- **Normalization** with PairNorm [10, Zaho, 2020].
- **Rewiring** Drop edges, at random [7, Rong, 2019] or in finding the potential good ones [3, Giraldo, 2023]

Over-smoothing and over-squashing are intrinsically related

Inevitable trade-off between these two issues, as they cannot be alleviated simultaneously.
Quadratic complexity in the number of nodes (or edges).

Key technical novelty of the paper

Theoretical Analysis

Delaunay rewiring

Is an extreme **4 steps rewiring** method.

1. First GNN[a] constructs **node embeddings** .
2. Reduce the embedding with **UMAP** in dim 2.
3. **Rebuilt edges with Delaunay triangulation**.
4. Second GNN **mix with the original features** of the graph.

[a] GCN from [4, Kipf and Welling, 2017]

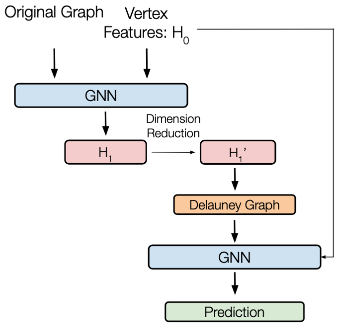


Figure 2: Illustration of the rewiring method using the features obtained by a GNN.

Initial Thoughts

Simplicity of the Method

No hyper-parameters = no grid-search.
Complexity of $\mathcal{O}(N \log N)$

Graph creation method

Create a graph from the embedding $\Rightarrow$ no need for the original graph.

Umap in 2 dimensions only

Triangulation in higher dimensions $\Rightarrow$ longer time + denser resulting graphs [a] + worse accuracy.

First GNN

Embed the initial smoothing and squashing? But needed for quality of embedding. Long range dependencies?

[a] Generalized triangles in dim=3: have 6 edges, 10 in dim=4

Delaunay Graph Properties

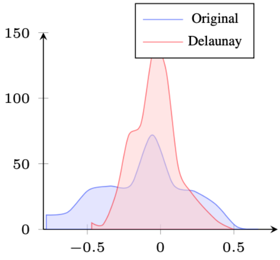
Sparse graphs: 6 times less edges (save computation time).
Raise the homophily value of heterophilic graphs.

Reduce over-squashing

$\iff$ Reduce high negative curved edges
$\iff$ maximize triangles + minimize squares.

Reduce over-smoothing

$\iff$ Reduce high positive curved edges.
Largest cliques limited to 3 nodes $\Rightarrow$ no over-smoothing [5, Nguyen et al, 2023].



(c): Cornell :
D_1= -0.18 D_9 =0.20
D_1= -0.49 D_9 =0.33

Figure: Effect of Delaunay rewiring on curvature distribution [Attali al., 2024] [2]

Experimental Evaluation

Methodology

Aim: Reproduce the rewiring experiment on the **Wisconsin dataset**[1].

Experimental setup

- **Device**: CUDA-enabled GPU with `PyTorch Geometric`, `UMAP`, `NetworkX`, `GraphRicciCurvature`
- **Preprocessing**: Feature normalization.
- **Runs**: 10 per experiment, max 2000 epochs, early stopping patience 100 epochs.

Key results

- **GCN** accuracy improved from 54.90% to 67.55% (+12.6%)
- **GAT** accuracy improved from 55.88% to 69.12% (+13.2%)
- Graph **homophily** increased by 96% (0.366 $\rightarrow$ 0.718)

$p \leq 0.0001$: statistically significant.

Significant performance gains across different model architectures. $\Rightarrow$ **Success**!

[1] From WebKB dataset, 251 nodes = web pages from Wisconsin connected by edges = hyperlinks, node features = bag-of-words in dim 1703, labels = 5 kind of author.

Results: Graph Property Analysis

- **Baseline Graph**
  - Mean Degree: 5.59
  - Homophily: 0.366
  - Curvature Range: [-0.475, 0.250]
- **Delaunay Graph**
  - Mean Degree: 7.83-7.87
  - Homophily: 0.704-0.718 (improved by 96%)
  - Curvature Range: [-0.214, 0.200]

	Original Graph	**DR**
Homophily	0.06	0.65
Number of edges	499	1470
Max degree	24	14
Mean degree	12	6
Accuracy GCN	55.12±1.51	**70.98±1.5**
Accuracy GAT	46.05 ±1.49	**74.33 ±1.24**
Time for triangulation (in sec)	-	≤ 1

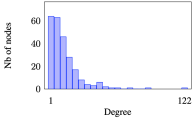


Figure 8: Histogram of the degree distribution for the original Wisconsin graph

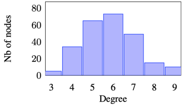
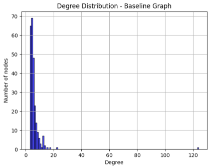


Figure 9: Histogram of the degree distribution for the Delaunay Wisconsin graph



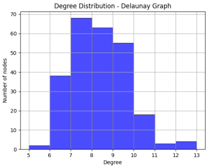
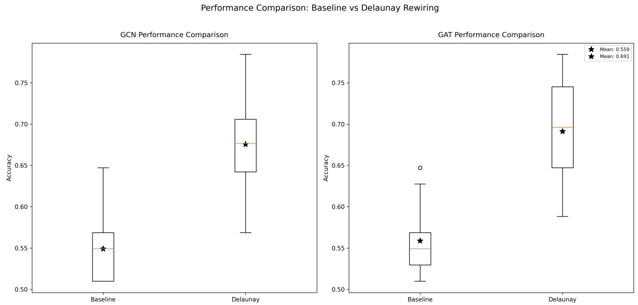


Figure: Effect of the Delaunay rewiring on degree distribution. Left: original, Right: after rewiring, Top: [Attali al., 2024] [2], Bottom: ours.

Results: Performance Improvements on Prediction Task



Performance Improvements

- **GCN**: 54.90% to 67.55% (+12.6%)
- **GAT**: 55.88% to 69.12% (+13.2%)
- **Statistical significance**: t-statistic:-8, $p \leq 0.0001$

Discussion

Performance

Delaunay rewiring **increase graph homophily** and **reduce negative curvature**, with more **balanced degree distribution**. Improvements are statistically significant (p $<$ 0.0001). GAT slightly outperformed GCN in both baseline and Delaunay settings.

Consistency of results

Delaunay graph properties show small variations, indicating stability. Performance improvements are robust across different random splits.

Limitations

- **Dimensionality reduction** loss of feature expression. We did not explore higher dimensions.
- **Computational considerations** Complexity of $\mathcal{O}(N \log N)$ only, but graph fully loaded into memory and UMAP + curvature computation.
- **Parameters**
  - UMAP has hyperparameters.
  - Dependence on feature normalization?
  - Effect of different data splits?

Conclusion

Conclusion

Findings

- We have understood the problem of over-smoothing and over-squashing
- We have understood the process from the authors.
- We were able to reproduce the experiment on the Wisconsin dataset.
- We confirm the results of the authors.

Future paper that will be explored in the report:

- *Cayley Graph Propagation* by JJ Wilson, Maya Bechler-Speicher, Petar Veličković [9]

Do you have any question?

References I

Uri Alon and Eran Yahav.
On the bottleneck of graph neural networks and its practical implications, 2021.

Hugo Attali, Davide Buscaldi, and Nathalie Pernelle.
Delaunay graph: Addressing over-squashing and over-smoothing using delaunay triangulation.
In *Forty-first International Conference on Machine Learning*, 2024.

Jhony H. Giraldo, Konstantinos Skianis, Thierry Bouwmans, and Fragkiskos D. Malliaros.
On the trade-off between over-smoothing and over-squashing in deep graph neural networks.
In *Proceedings of the 32nd ACM International Conference on Information and Knowledge Management*, CIKM '23, page 566–576. ACM, October 2023.

T. N. Kipf and M. Welling.
Semi-supervised classification with graph convolutional networks.
In *Proceedings of the International Conference on Learning Representations*, 2017.

K. Nguyen, N. M. Hieu, V. D. Nguyen, N. Ho, S. Osher, and T. M. Nguyen.
Revisiting over-smoothing and over-squashing using ollivier-ricci curvature.
In *International Conference on Machine Learning*, pages 25956–25979. PMLR, 2023.

Chien-Chun Ni, Yu-Yao Lin, Jie Gao, Xianfeng David Gu, and Emil Saucan.
Ricci curvature of the internet topology, 2015.

References II

Y. Rong, W. Huang, T. Xu, and J. Huang.
Dropedge: Towards deep graph convolutional networks on node classification.
In *International Conference on Learning Representations*, 2019.

Jake Topping, Francesco Di Giovanni, Benjamin Paul Chamberlain, Xiaowen Dong, and Michael M. Bronstein.
Understanding over-squashing and bottlenecks on graphs via curvature, 2022.

JJ Wilson, Maya Bechler-Speicher, and Petar Veličković.
Cayley graph propagation, 2024.

Lingxiao Zhao and Leman Akoglu.
Pairnorm: Tackling oversmoothing in gnns.
In *International Conference on Learning Representations*, 2020.

Curvature

Paper: Balance Forman Curvature [8, Topping, 2022] is computed over cycles of size 4.
Experiment: Oliver-Ricci Curvature [6, Ni, 2015] `GraphRicciCurvature.OllivierRicci`.

$$c_{ij} = \frac{2}{d_i} + \frac{2}{d_j} - 2 + 2\frac{\sharp_\Delta}{\max(d_i, d_j)} + \frac{\sharp_\Delta}{\min(d_i, d_j)} + \frac{\max(\sharp_\square^i, \sharp_\square^j)^{-1}}{\max(d_i, d_j)}(\sharp_\square^i + \sharp_\square^j)$$

where $\sharp_\Delta$ is the number of triangles based at e_{ij}, $\sharp_\square^i$ is the number of 4-cycles based at e_{ij} starting from i without diagonals inside.

Curvature of graph edges

- Positive curvature edges establish connections between nodes belonging to the same community.
  Highly positive curved edges $\rightarrow$ over-smoothing [5, Nguyen et al., 2023].
- Negative curvature edges connect nodes from different communities.
  Highly negative curved edges $\rightarrow$ over-squashing [8, Topping et al., 2021].

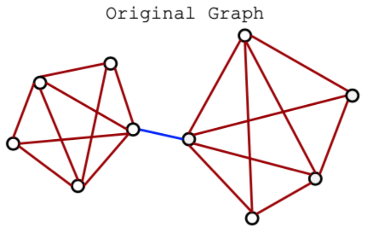


Figure: Example graph: in red the edges with positive curvature ($\sim$ 3), in blue with negative curvature (-1.2) [2, Attali al., 2024]

Delaunay Triangulation

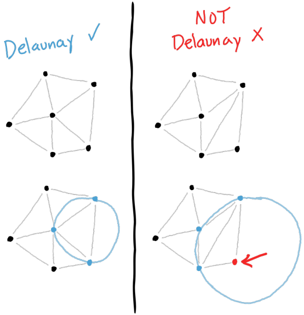
Definition

A Delaunay triangulation, denoted as $DT(P)$, for a set P of points in the d-dimensional Euclidean space, is a triangulation where no point in P resides within the circum-hypersphere of any d-simplex in $DT(P)$.

- **Experiment function:** We use the SciPy implementation with the *joggled input* parameter.

  `scipy.spatial.Delaunay(positions, qhull_options=QJ)`

- **Geometric interpretation:** In two dimensions, Delaunay triangulations maximize the angles of triangles formed by a set of points $\rightarrow$ triangle $\sim$ equilateral. *Figure: Sam Westrick*



UMAP

Uniform Manifold Approximation and Projection (UMAP) is a dimensionality reduction technique that can be used for visualisation similarly to t-SNE, but also for general non-linear dimension reduction. UMAP constructs a high dimensional graph representation of the data then optimizes a low-dimensional graph to be as structurally similar as possible.

Advantages

- **Speed**: UMAP is faster than t-SNE.
- **Global structure**: UMAP preserves more of the global structure.
- **Separation**: clearly separate groups of similar categories.

Dimensionality reduction technique is not perfect - by necessity, we're distorting the data to fit it into lower dimensions - and UMAP is no exception. But it is a powerful tool to visualize and understand large, high-dimensional datasets.

Hyperparameters choice

Most common: `n_neighbors` and `min_dist`, control the balance between local and global structure.

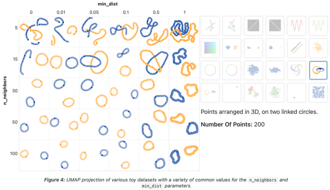


Figure: Illustration of UMAP hyperparameters from Google PAIR

Graph Neural Networks

GCN

- Hidden channels: 32
- Two layers with ReLU activation
- Dropout: 0.5
- Learning rate: 0.005
- Weight decay: 5e-6

GCN

- Hidden channels: 32
- First layer: 8 attention heads
- Second layer: 1 attention head
- Dropout: 0.5
- Learning rate: 0.005
- Weight decay: 5e-6

Runtime Performance

Preprocessing Time:

- UMAP dimensionality reduction: 1-2 seconds
- Delaunay triangulation: ¡ 1 second
- Curvature calculation: 3-5 seconds per graph
- Total preprocessing overhead: 5-8 seconds

Training Performance:

- Average epochs until convergence:
  - Baseline GCN: 150 epochs
  - Delaunay GCN: 130 epochs
  - Baseline GAT: 180 epochs
  - Delaunay GAT: 160 epochs
- Training time per epoch:
  - GCN: 0.1 seconds
  - GAT: 0.2 seconds
- Total training time per run:
  - Baseline models: 15-35 seconds
  - Delaunay models: 13-32 seconds

Memory Usage:

- Peak memory during preprocessing: 2GB
- Training memory footprint:
  - Baseline: 1GB
  - Delaunay: 1.2GB
- Additional storage for results: ¡ 100MB