Delaunay Rewiring to Avoid Over-Squashing and Over-Smoothing in Graphs

Edwin Roussin [*1] **Tristan Waddington** [*1]

Abstract

This document reviews the graph rewiring method proposed by (Attali et al., 2024) based on Delauynay triangulation that aims to avoid over-smoothing and over-squashing during prediction tasks on a graph. It uses notions of edge curvature to measure the quality of the rewiring. The method is tested on a variety of graph datasets and compared to other ones by the authors. We have reproduced some experiments with slightly lesser results. Ultimately, we have gotten better accuracy results with the Cayley Graph Propagation method proposed by (Wilson et al., 2024).

1. Delaunay Rewiring

Graph neural networks (GNNs) have emerged as the standard approach for effective learning graph-structured data. GNNs employ an iterative approach, updating node representations through the local aggregation of information from neighboring nodes, known as the message-passing paradigm (Gilmer et al., 2017). However, this iterative process can lead to over-smoothing and over-squashing, where the node representations become too similar to each other, or ineffective to transmit long-range information. In these cases, the model performance can degrade.

1.1. Over-Smoothing and Over-Squashing

Over-Smoothing Message-passing neural networks (MPNN) use an iterative approach, updating node representations through the local aggregation of information from neighboring nodes. The need to stack additional layers to capture non-local interactions tends to make node features more and more similar, leading to over-smoothing. **This is particularly problematic when the graph is heterophilic**, i.e., when nodes belong to different communities (Zheng et al., 2022).

[*]Equal contribution [1]Institut Polytechnique de Paris, Palaiseau, France. Correspondence to: <first-name.lastname@polytechnique.edu>.

Final Exam for APM_5DS30_TP, Machine Learning with Graphs, IPP, 2025.

Over-Squashing The squashing effect occurs when the model is unable to transmit long-range information, leading to a loss of information. MPNN models try to capture exponentially growing information into fixed sized representations. (Alon & Yahav, 2021) have shown the **correlation between overs-quashing and bottleneck in the graph structure**. Models such as Graph Convolutional Networks (GCN) are known to suffer from this issue because they absorb the information from all edges equally.

1.2. Edge Curvature

The main metrics used to characterize graph structure and measure the quality of the rewiring is the discrete Ricci edge curvature. A complete definition can be found on Appendix A.2. Previous work has shown that:

- Positive curvature edges establish connections between nodes belonging to the same community. Highly positive curved edges cause over-smoothing (Nguyen et al., 2023).
- Negative curvature edges connect nodes from different communities. Highly negative curved edges cause over-squashing (Topping et al., 2022).

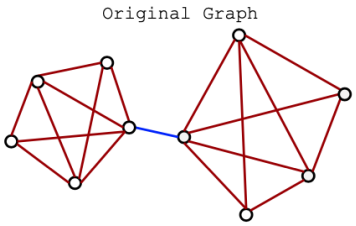

Figure 1. Example graph: in red the edges with positive curvature (~ 3), in blue with negative curvature (-1.2) (Attali et al., 2024)

Hence, we get a simple metric for further experiments where we expect to see the edge curvature amplitude decrease after the rewiring process.

1.3. Rewiring

1.3.1. Former methods' limitation

Existing methods mitigate over-squashing by rewiring the input graph to minimize structural bottlenecks. First ones rely on the analysis of the graph structure, through local or

global features like edge curvature or resistance. However, these methods may not scale well with the number of nodes and depend on the choice of hyperparameters. Moreover, they need an original graph to modify, which is not always available in some applications.

Conversely, over-smoothing is avoided by preventing embedding to become the same through: **Normalization** with PairNorm (Zhao & Akoglu, 2020); or **rewiring**, in dropping edges, at random (Rong et al., 2019) or in finding the potential good ones (Giraldo et al., 2023)

1.3.2. Delaunay Triangulation

Delaunay rewiring Is an extreme **4 steps rewiring** method illustrated bellow and detailed in Algorithm 1:

1. Construct the **node embeddings** from the original graph with a first GNN[1].
2. Reduce the embeddings with **UMAP** in dim 2.
3. **Rebuild edges with Delaunay triangulation** from their distance in UMAP embedding space.
4. Predict with a second GNN on the **mix of egdes from Delaunay graph with the original features**.

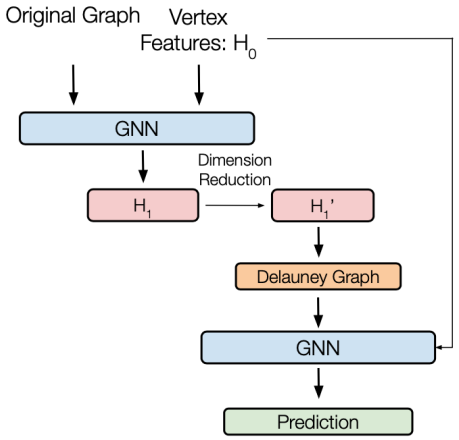

Figure 2. Illustration of the Delaunay [Attali al., 2024] (Attali et al., 2024)

GNN embedding Initially, the authors triangulated the graphs considering the original features. However, these starting features often lack expressiveness or suffer from poor quality. So they used the strategy to add the first step of GNN embedding - as pictured above - to improve the quality of the Delaunay graph. In our following experiment, the initial embedding is not used as the data is good enough. We did not experiment this embedding.

Initial thoughts The method is remarkably simple, as it does not require any hyperparameters according to authors, which eliminates the need for a grid search. Its computational complexity is efficient, scaling as $\mathcal{O}\big(N \log N\big)$. Additionally, the method constructs a graph directly from the embedding, making it independent of the presence of the original graph.

[1]GCN from (Kipf & Welling, 2017)

Algorithm 1 Delaunay Rewiring

Input: dataset x, graph $G = (H_0, E)$
Compute node embeddings H_1 with GNN1 from G {if needed}
Reduce to 2D with UMAP $H_1 \rightarrow H_2$
Compute Delaunay triangulation $DT(H_2)$
$E' = \emptyset$
for $(i, j) \in DT(H_2).simplices$ **do**
  $E' = E' \cup \{(i, j)\}$ {Add undirected edge}
end for
$G' = (H_2, E')$ {Rewired graph}
Compute node embeddings H' with GNN2 from $G'' = (H_0, H_2, E')$ **Return:** predicted class of nodes

The use of UMAP is restricted to two dimensions. This is because performing triangulation in higher dimensions increases computation time and results in denser graphs[2]. This increased density reduced the accuracy in experiments, making the two-dimensional approach more practical and effective.

Authors have added the first GNN late in their research as the Delaunay Graph need quality embedding. However, it raises questions about whether these initial representations are victims of smoothing and squashing effects by this GNN. Does this GNN effectively captures long-range dependencies?

1.4. Experiments

We have reproduced the rewiring experiment on the **Wisconsin dataset**[3]. The report in Table 1 demonstrate substantial improvements in graph neural network performance. The effect of the Delaunay rewiring is clearly visible on the degree distribution in the Figure 3 below. The curvature distribution of the edges is also shown in the Figure 4.

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% $(0.366 \rightarrow 0.718)$. All improvements are statistically significant ($p < 0.0001$).

Impact: The Delaunay rewiring approach successfully ad-

[2]Generalized triangles in three dimensions have six edges, while in four dimensions, they have ten edges.

[3]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.

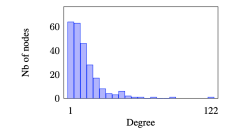

Figure 8: Histogram of the degree distribution for the original Wisconsin graph

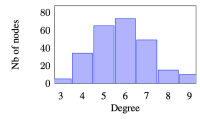

Figure 9: Histogram of the degree distribution for the Delaunay Wisconsin graph

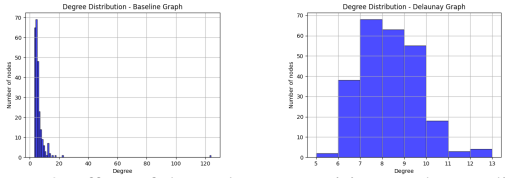

Figure 3. Effect of the Delaunay rewiring on degree distribution. Left: original, Right: after rewiring, Top: [Attali al., 2024] (Attali et al., 2024), Bottom: ours.

dresses over-squashing and improves graph structure, leading to significant performance gains across different model architectures.

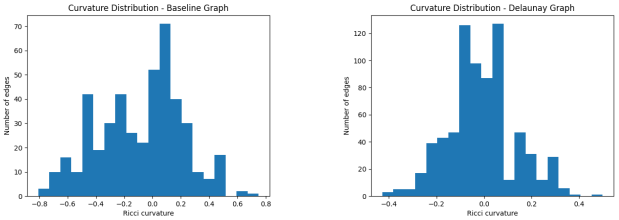

Figure 4. Evolution of the curvature distribution of nodes after rewiring (right). We can confirm that the amplitude of the curvature has decreased.

Validation: Results are robust across multiple experiments and statistically significant, with comprehensive analysis of graph properties supporting the improvements.

1.5. Limitations

Dimensionality Reduction We might lose some feature information during the UMAP reduction to 2 dimensions. The quality of the Delaunay graph depends entirely on the quality of reduced features.

Computational Considerations The method complexity is in $\mathcal{O}(N \log N)$ only, but the two graphs are fully loaded into memory. But this is way better than the quadratic cost of other methods that are enumerated in the original paper.

Table 1. Comparison of Baseline and Delaunay Graph Metrics

Metric	Orignal	Delaunay Graph
Mean Degree	5.59	7.85
Homophily	0.366	0.710 (↑ 96%)
Curvature Range	[-0.475, 0.250]	[-0.214, 0.200]

Parameter Sensitivity **We do not entirely agree with the authors as we find out that the impact of UMAP parameters not fully explored.** They could modulate the quality of the embedding we already discussed. Moreover, the potential dependence on feature normalization and the effect of different train/val/test splits is not extensively studied.

1.6. Future Work

If we were to work further in this topic, we could focus on:

- **Algorithmic Improvements**: Investigate higher-dimensional Delaunay triangulation, explore sparse approximations for larger graphs, develop incremental/streaming versions for large-scale graphs or optimize UMAP parameter selection.
- **Analysis Extensions**: Study impact on different graph properties, investigate relationship between feature space and graph structure, compare with other rewiring methods on the same dataset, or analyze feature importance in graph construction.
- **Ablation Studies**: Compare with other dimensionality reduction method, analyze the effect of different feature preprocessing, the sensitivity to hyperparameters and try different triangulation algorithms.

2. Additional Paper

We have chosen to delve into another rewiring method proposed by (Wilson et al., 2024) based on Cayley graphs, named **Cayley Graph Propagation (CGP)**. The authors have focused on the creation of bottleneck-free structure to ameliorate the over-squashing effect.

2.1. Choice explanation

We selected this paper because it aligns with the concept of **totally rewiring the initial graph using a mathematical structure**. On one hand, Delaunay Triangulation leverages geometric properties. On the other hand, the Cayley graph is a mathematical structure that can represent a group. We believe these two methods are both competitive and complementary. Notably, both papers were published in 2024 and do not reference each other.

2.2. Paper summary

(Wilson et al., 2024) **precompute bottleneck-free graph structures** to alleviate the over-squashing effect. They propose the use of a well-known **expander graph** family: the Cayley graphs of the $\mathrm{SL}(2, \mathbb{Z}_n)$ special linear group[4] as a

[4]In mathematics, the special linear group $SL(n, R)$ of degree n over a commutative ring R is the set of $n \times n$ matrices with

computational template for GNNs.

They have improved previous work that already used this family, named Expander Graph propagation (EGP), that required to truncate the graph to align with the input graph. The authors show that truncation is not necessary and that the complete Cayley graph can be used directly with the direct effect of reducing the diameter of the graph as illustrated in the Figure 5. They have named their new method Cayley Graph Propagation (CGP).

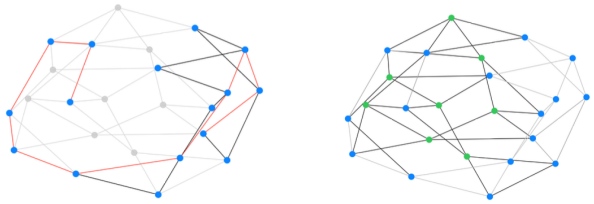

Figure 5. Both Cayley graphs represent $SL(2, \mathbb{Z}_3)$ with $|V| = 24$ nodes using the same construction. Left: A truncated Cayley graph (spectral gap: 0.0751, diameter: 10) aligned to a given input graph. Right: The complete Cayley graph (spectral gap: 1.2679, diameter: 4) structure indicating the additional virtual nodes (in green). Source: (Wilson et al., 2024)

The authors have shown that the spectral gap[5] of the Cayley graph is a good indicator of the quality of the graph structure. A larger spectral gap defines a strong connectivity, or alternatively the global lack of bottlenecks.

2.3. Comparison

The constructed Cayley graph with $|V|$ nodes has a low diameter, requiring only $\mathcal{O}(log(|V(G)|)$ steps to globally propagate information. This enhances its ability to eliminate over-squashing and bottlenecks, which is further supported by having a higher spectral gap.

But this construction requires adding new nodes into the graph; hence, we need to modify the feature matrix into an extended version, $\mathbf{X}' \in \mathbb{R}^{(|V|+virt)|\times d}$. The first $|V|$ nodes are featured using the data from $\mathbf{X}$, the vitrual ones are initialized as zeros.

Furthermore, this new method does not suffer from limitations of the Cayley graph construction: the inability to find the best way to align it to a given input graph, mitigating the potential for stochastic effects in the process. The additional virtual nodes act as "bridges" between poorly connected communities in the Cayley graph, ameliorating any poorly-connected regions caused by misalignment: see how the former red string in now connected in the Figure 5. However, the absence of curvature metrics in the Cayley paper makes it difficult for us to directly compare the two methods on the over-smoothing factor.

Hence, to compare these methods, we evaluated both approaches on the MUTAG dataset (Debnath et al., 1991) a binary graph classification task containing 188 molecular graphs. Using a 4-layer GIN architecture with batch normalization and 5-fold stratified cross-validation, we assessed each method's ability to preserve and leverage molecular graph structure. While CGP achieved strong performance ($92.01\% \pm 3.80\%$), Delaunay rewiring showed lower accuracy ($88.31\% \pm 2.64\%$), likely due to information loss during the dimensional reduction step or the non-use of a first GNN. Full results are shown in Table 2 and more details can be found on Appendix C. **This suggests that preserving graph topology through virtual nodes, as in CGP, might be more effective than complete geometric rewiring for molecular property prediction tasks.**

Table 2. Comparison of Delaunay and Cayley methods on MUTAG (5 folds).

	Orignal	**Delaunay**	**CGP**	**EGP**
Accuracy	89.84%	88.31%	92.01%	89.90%
Variance	$\pm 5.05\%$	$\pm 2.64\%$	$\pm 3.80\ \%$	$\pm 1.94\%$

Software and Data

The original code repository and additional work can be found on GitHub[6]. The code is written in Python and uses the PyTorch library. The code is available under the MIT license.

Acknowledgements

We thank our professor Mr. Jhony H. Giraldo for presenting us this article and the theoretical foundations to understand it.

Impact Statement

This paper highlight the promising yet simple method of Delauynay Rewiring to improve the performance of graph-based machine learning models. While promising, we encourage further experiments on more ambitious datasets.

determinant 1, with the group operations of ordinary matrix multiplication and matrix inversion. (Wikipedia)

[5]The spectral gap is the difference between the first and second normalized eigenvalues of the graph Laplacian and was not computed in our experiments.

[6]https://github.com/waddason/Delaunay-Rewiring

References

Alon, U. and Yahav, E. On the bottleneck of graph neural networks and its practical implications. In *International Conference on Learning Representations*, 2021.

Attali, H., Buscaldi, D., and Pernelle, N. Delaunay Graph: Addressing Over-Squashing and Over-Smoothing Using Delaunay Triangulation. In *Forty-first International Conference on Machine Learning*, 2024. URL https://openreview.net/forum?id=uyhjKoaIQa.

Debnath, A. K., Lopez de Compadre, R. L., Debnath, G., Shusterman, A. J., and Hansch, C. Structure-activity relationship of mutagenic aromatic and heteroaromatic nitro compounds. correlation with molecular orbital energies and hydrophobicity. *Journal of Medicinal Chemistry*, 34 (2):786–797, 1991. doi: 10.1021/jm00106a046. URL https://doi.org/10.1021/jm00106a046.

Gilmer, J., Schoenholz, S. S., Riley, P. F., Vinyals, O., and Dahl, G. E. Neural message passing for quantum chemistry. In *International Conference on Machine Learning*, pp. 1263–1272. PMLR, 2017.

Giraldo, J. H., Skianis, K., Bouwmans, T., and Malliaros, F. D. 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, pp. 566–576. ACM, October 2023. doi: 10.1145/3583780.3614997. URL http://dx.doi.org/10.1145/3583780.3614997.

Huang, Y., Zeng, Y., Wu, Q., and Lü, L. Higher-order graph convolutional network with flower-petals laplacians on simplicial complexes, 2024. URL https://arxiv.org/abs/2309.12971.

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

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

Ni, C.-C., Lin, Y.-Y., Gao, J., Gu, X. D., and Saucan, E. Ricci curvature of the internet topology, 2015. URL https://arxiv.org/abs/1501.04138.

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

Topping, J., Giovanni, F. D., Chamberlain, B. P., Dong, X., and Bronstein, M. M. Understanding over-squashing and bottlenecks on graphs via curvature, 2022. URL https://arxiv.org/abs/2111.14522.

Wilson, J., Bechler-Speicher, M., and Veličković, P. Cayley graph propagation, 2024. URL https://arxiv.org/abs/2410.03424.

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

Zheng, X., Liu, Y., Pan, S., Zhang, M., Jin, D., and Yu, P. S. Graph neural networks for graphs with heterophily: A survey. *arXiv preprint arXiv:2202.07082*, 2022.

A. Additional details on the Delaunay Rewiring method

A.1. Delaunay Triangulation

The Delaunay triangulation is a method to construct a graph from a set of points in a space. It is a well-known method in computational geometry and has been used in various applications.

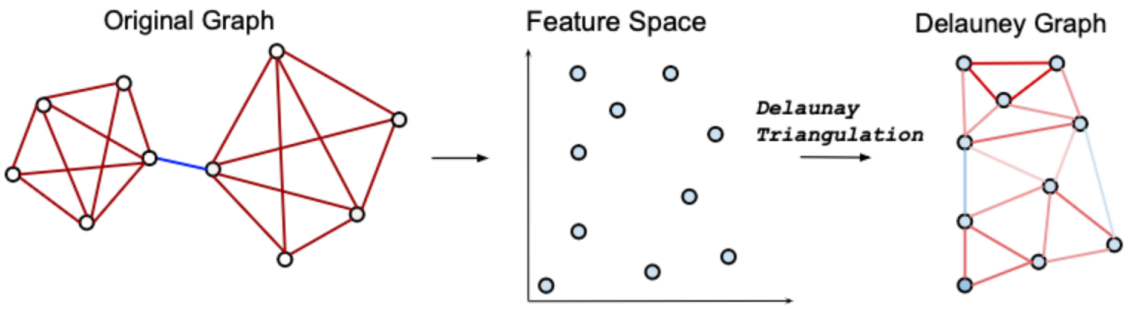

Figure 6. Illustration of the Delaunay rewiring process in a 2-dimension feature space.

Definition A.1. 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 $DT(P)$.

A.2. Edge Curvature

We describe bellow the two main curvature metrics used in the Delaunay rewiring method.

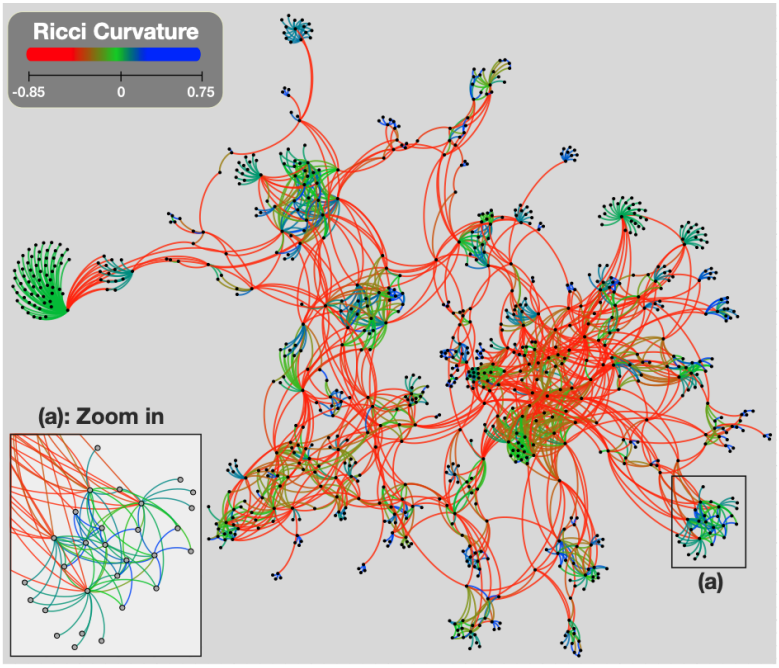

Figure 7. It shows the Ricci curvature of each edge in a router level graph (Exodus(US)) from the Rocketfuel data set with 895 nodes and 2071 edges. Negatively curved edges (in red) behave like "backbones", maintaining the connectivity of clusters that are grouped by zero and positively curved edges (in green and blue). Source (Ni et al., 2015)

Paper : Balance Forman Curvature (Topping et al., 2022) is computed over cycles of size 4.

Balance Forman Curvature:

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

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

Experiment : **Oliver-Ricci Curvature** (Ni et al., 2015)

We used the specific implementation from `GraphRicciCurvature.OllivierRicci`. Node Ricci curvature is defined as the average of all its adjacency edge. A visual representation of the curvature of edges in a graph is shown in the 7 below from the original paper.

A.3. UMAP

Method presentation Uniform Manifold Approximation and Projection (UMAP) is a dimensionality reduction technique that can be used for visualization 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. These methods present great advantages:

- **Speed**: UMAP is faster than t-SNE.
- **Global structure**: UMAP preserves more of the global structure.
- **Separation**: UMAP 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. They have a significant impact on the resulting embedding, and the choice of these parameters is crucial. A visual example of their effect on the same two linked circles is shown in the Figure 8 below.

- `n_neighbors`: number of neighbors used to construct the high-dimensional graph.
- `min_dist`: minimum distance between points in the low-dimensional space.

B. Experiment Details on the Delaunay Rewiring method

B.1. Wisconsin Dataset

WebKB is a dataset that includes web pages from computer science departments of various universities. It represents $4,518$ web pages that are categorized into 6 imbalanced categories (Student, Faculty, Staff, Department, Course, Project). Additionally there is Other miscellanea category that is not comparable to the rest. We have used the Wisconsin subset of the dataset, which contains 251 web pages from the University of Wisconsin-Madison.

We used the classical dataset splitting from Train/Val/Test Split: $60\%/20\%/20\%$ as proposed by (Attali et al., 2024). We add our own data preprocessing in normalizing the features. Our experiment modestly reaches a 69.12% accuracy on the Wisconsin dataset, which has been outperformed by other methods since 2019 has shown in the leaderboard bellow. [7]

The results obtained by (Attali et al., 2024) are summarized in the Table 3 below.

B.2. Graph Neural Networks

We used two popular GNN architectures: Graph Convolutional Networks (GCN) and Graph Attention Networks (GAT).

[7] Highest accuracy on the Wisconsin dataset is 90.5% by (Huang et al., 2024), using Higher-order Graph Convolutional Network (HiGCN) grounded in Flower Petals Laplacians, capable of discerning intrinsic features across varying topological scales.

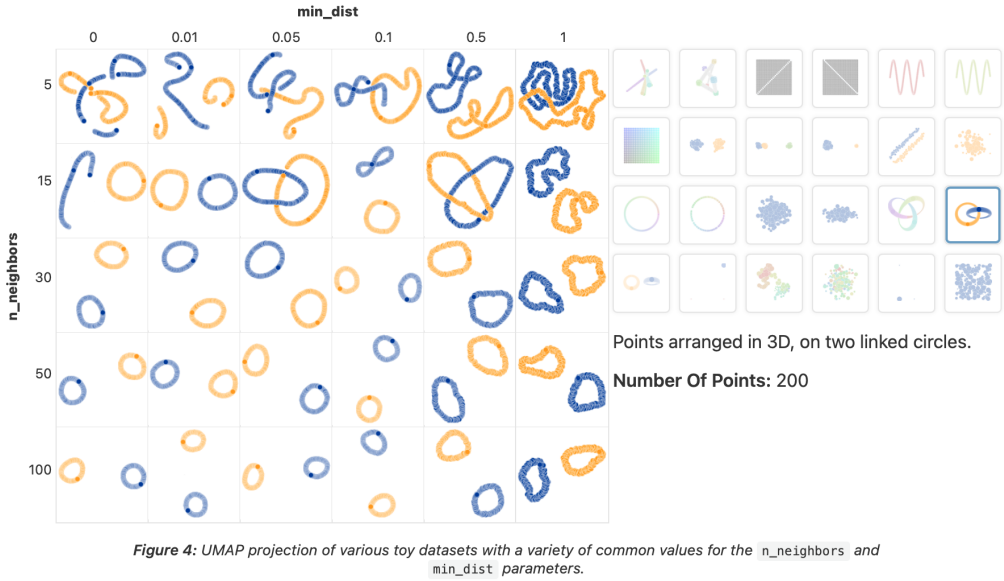

Figure 4: *UMAP projection of various toy datasets with a variety of common values for the* `n_neighbors` *and* `min_dist` *parameters.*

Figure 8. Illustration of UMAP hyperparameters from Google PAIR

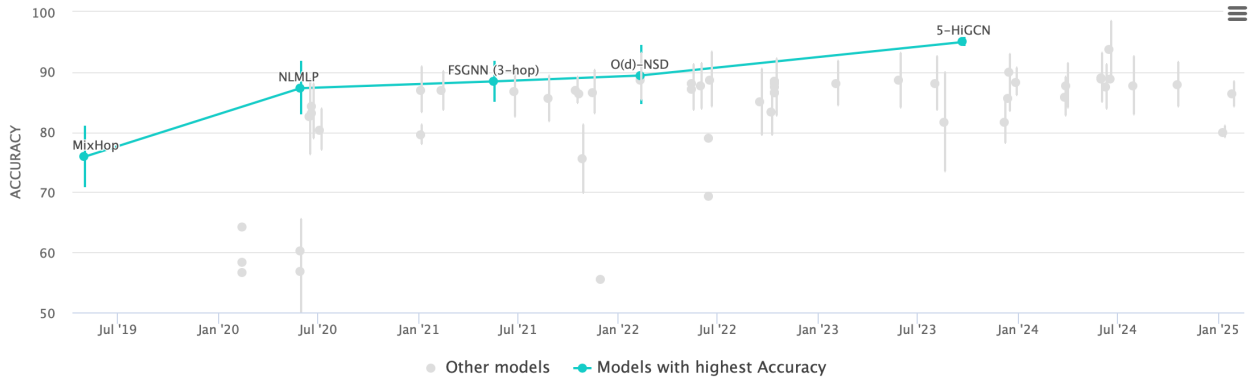

Figure 9. Performance of GNN on the Wisconsin dataset, from Paper With Code.

Table 3. Experimental results with Delaunay Rewiring in different dimensions reduction for Wisconsin dataset

Metric	Original	DR (dim=2)	DR (dim=3)	DR (dim=4)	DR (dim=5)	DR (dim=7)	Ours
Homophily	0.06	0.65	0.63	0.60	0.54	0.44	0.71
Number of edges	499	1470	2534	4064	6266	16148	-
Max degree	24	14	24	42	57	167	14
Mean degree	12	6	12	18	28	66	8
Accuracy GCN (%)	55.12 ± 1.51	**70.98 ± 1.5**	69.45 ± 1.5	68.59 ± 1.5	68.55 ± 1.5	66.42 ± 1.7	67.55
Accuracy GAT (%)	46.05 ± 1.49	**74.33 ± 1.24**	74.23 ± 1.4	70.75 ± 1.4	72.43 ± 1.5	67.16 ± 1.7	69.12
Triangulation time (s)	-	≤ 1	≤ 1	≤ 1	2	50	≤ 1

B.3. Experimental setup

Hardware and Software The experiments were conducted on a CUDA-enabled GPU with the following software dependencies:

Table 4. Hyperparameters for GCN and GAT Models

Hyperparameter	**GCN**	**GAT**
Hidden Channels	32	32
Layers	2 (ReLU activation)	2 (8 attention heads in 1st layer, 1 in 2nd)
Dropout	0.5	0.5
Learning Rate	0.005	0.005
Weight Decay	5×10^{-6}	5×10^{-6}
Training Time per Epoch	0.1s (empirical)	0.2s (empirical)

- **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.

Preprocessing Time:

- UMAP dimensionality reduction: 1-2 seconds.
- Delaunay triangulation: < 1 second.
- Curvature calculation: $\sim 3 - 5$ seconds per graph.
- Total preprocessing overhead: $\sim 5 - 8$ seconds.

Table 5. Training Metrics Comparison

Average Epochs Until Convergence	**Baseline**	**Delaunay**	**s/epoch**
GCN	$\sim$ 150 epochs	$\sim$ 130 epochs	0.1s
GAT	$\sim$ 180 epochs	$\sim$ 160 epochs	0.2s

Memory Usage Peak memory during preprocessing: $\sim 2GB$, Training memory footprint: Baseline: $\sim$ $1GB$; Delaunay: $\sim 1.2GB$. Additional storage for results: $<$ 100MB.

B.4. Results

Our experiment modestly reaches a 69.12% accuracy on the Wisconsin dataset, which has been outperformed by other methods since 2019 has shown in the leaderboard Figure 9. The results are summarized in Table 6.

Table 6. Comparison of Baseline and Delaunay Graph Metrics

Metric	**Orignal**	**Delaunay Graph**
Mean Degree	5.59	7.85
Homophily	0.366	0.710 ($\uparrow$ 96%)
Curvature Range	[-0.475, 0.250]	[-0.214, 0.200]
GCN accuracy	54.90%	67.55% ($\uparrow$ 12.6%)
GAT accuracy	55.88%	69.12% ($\uparrow$ 13.2%)

Conclusion The Delaunay rewiring approach demonstrates significant and consistent improvements on the Wisconsin dataset. The improvements are not only substantial in magnitude $(12.6 - 13.2\%)$ but also statistically significant, with both GCN and GAT models benefiting from the rewiring. The enhanced graph properties (improved homophily and reduced negative curvature) provide structural evidence for why the approach works well. While there are some limitations and areas for future investigation, the current results strongly support the effectiveness of this approach for improving graph neural network performance.

C. Experiment Details on the Cayley Graph Propagation method

C.1. MUTAG Dataset

The MUTAG dataset (Debnath et al., 1991) is a benchmark dataset for graph classification tasks. It consists of 188 molecular graphs, each representing a chemical compound. The task is to classify these graphs into two categories: mutagenic and non-mutagenic compounds. Each graph is represented by its adjacency matrix, and the node features are derived from the chemical properties of the atoms in the molecules. The dataset is relatively small, making it suitable for testing new graph neural network architectures and methods. The graphs in the MUTAG dataset are relatively small, with an average of 18 nodes and 19 edges per graph. The dataset is imbalanced, with 87 mutagenic and 101 non-mutagenic graphs.

The dataset is available in the `PyTorch Geometric` library, and it can be easily loaded using the `torch_geometric.datasets` module. The dataset is commonly used in the graph neural network community for benchmarking and evaluating new models and methods.

C.2. Experimental setup

This dataset is used to compare the Delaunay rewiring method with the Cayley Graph Propagation method. For statistical validity, we used a stratified 5-fold cross-validation approach to ensure that each fold contains a representative distribution of the classes and observe the variance of the results across the folds. The final results reported in Table 2 compare the same GIN-4 layer model on the graphs modified by each method : Original graph, Delaunay rewiring, EGP, and CGP.

- **Original graph**: The original graph without any modifications.
- **Delaunay rewiring**: The graph after applying the Delaunay triangulation method. It uses robust triangulation and add small random noise $(1e^{-6})$ to avoid co-planar points. The resulting graphs are undirected and include self-loops.
- **EGP**: The graph after applying the Cayley Graph Propagation method. It uses truncated version of the Cayley graph of the $\mathrm{SL}(2, \mathbb{Z}_n)$ special linear group. No virtual nodes are added.
- **CGP**: The graph after applying the Cayley Graph Propagation method. It uses the complete Cayley graph of the $\mathrm{SL}(2, \mathbb{Z}_n)$ special linear group. Virtual nodes are added to the graph to improve the connectivity and reduce the diameter. The virtual nodes are initialized with zeros features.

C.3. Implementation Details

The common model architecture is based on a Graph Isomorphism Network (GIN) with 4 layers, each having a hidden dimension of 64. The network applies a dropout rate of 0.5 to prevent overfitting, and uses global mean pooling for graph-level representation. Batch normalization is applied after each convolutional layer to stabilize and accelerate training. This setup ensures the model's expressive power and robustness for graph classification tasks.

The training configuration employs the Adam optimizer with a learning rate of 0.001, complemented by a ReduceLROnPlateau scheduler (`patience=20, factor=0.5`) for dynamic learning rate adjustment. A batch size of 32 is used, and early stopping with patience of 50 epochs aims to prevent overfitting. The maximum number of training epochs is capped at 200 to ensure timely completion of experiments. These settings balance computational efficiency and model performance.

For the Delaunay transform, UMAP is used to reduce high-dimensional node embeddings to 2D space, enabling Delaunay triangulation. UMAP parameters include `n_neighbors=5`, `min_dist=0.1`, and `metric=Euclidean`, with `n_components=2` for 2D projection. The triangulation process uses the QJ option for numerical stability, with small random noise added to node positions to avoid degenerates. The resulting graph is undirected and includes self-loops to maintain consistency with the original graph structure.

C.4. Results and Analysis

The CGP method achieves the highest accuracy among the evaluated approaches, although it exhibits moderate variance. EGP, on the other hand, provides the best stability with good accuracy, making it a reliable choice. Interestingly, the original method remains surprisingly competitive but suffers from instability in its results. The Delaunay method offers consistent performance but falls slightly short in terms of accuracy compared to the other methods.

All methods perform within a standard deviation of approximately $\pm 4.05\%$, with EGP demonstrating remarkably low

variance at $\pm 1.94\%$. The use of stratified k-fold cross-validation effectively reduces evaluation bias, ensuring a fair comparison across the methods.

Training dynamics reveal that early stopping typically occurs between 40 and 120 epochs, depending on the method. Learning rate reduction plays a crucial role in stabilizing training across all approaches, and careful validation patience settings further enhance the reliability of the results.

In conclusion, while CGP stands out for its accuracy, EGP's stability makes it a strong contender. The original method and Delaunay rewiring also show promise, with room for improvement in specific areas. These findings highlight the trade-offs between accuracy, stability, and consistency in graph-based learning methods.