An Experimental Look at the Stability of Graph Neural Networks against Topological Perturbations

The Relationship Between Graph Properties and Stability

Yigit Colakoglu

Responsible Professor: Elvin Isufi
Supervisors: Mohammad Sabbaqi, Maosheng Yang
EEMCS, Delft University of Technology, The Netherlands

A Thesis Submitted to EEMCS Faculty Delft University of Technology,
In Partial Fulfilment of the Requirements
For the Bachelor of Computer Science and Engineering
June 23, 2024

Name of the student: Yigit Colakoglu
Final project course: CSE3000 Research Project
Thesis committee: Elvin Isufi, Maosheng Yang, Mohammad Sabbaqi, Klaus Hildebrandt

Abstract

GNNs are a powerful tool for learning tasks on data with a graph structure. However, the topology of the graph in which GNNs are trained is often subject to change due to random, external perturbations. This research investigates the relationship between 5 topological properties of graphs (assortativity, density, edge connectivity, closeness centrality, diameter) and how stable GNNs trained on graphs with different topological properties are against different perturbations. The analysis is conducted by first synthetically generating graphs with different topological properties and training a GNN using the generated graphs. The synthetic graphs are then perturbed, and the relative change in the GNNs' output is measured. These results are further supported by conducting the same process on three popular GNN datasets: Cora, CiteSeer and PubMed citations. Finally, relationships between the graph properties under investigation and GNN stability are inferred using the results obtained from both synthetic and real-world datasets.

1 Introduction

Graphs are data structures that can be used to model interactions between entities with different features, using pairwise connections. They are commonly used in many fields, from the Internet of Things [20] to chemistry [8]. They are instrumental in signal processing because they can represent signals with complex structural information that cannot be represented in Euclidean space.

Their extensive modelling abilities led to the creation of techniques to analyze graph data that make use of not only the node features but also the graph's structural information [2]. For this task, Graph Neural Networks (GNN) were developed. Graph Neural Networks are very similar to Euclidean machine learning techniques in that they both try to minimize a given risk function by tuning the model's parameters. However, the main difference between them is that GNNs have an extra input: the structure, i.e., the graph's topology.

Even though the graph's topology provides valuable context for GNNs, they are subject to change. These minor alterations to a graph's original topology are called perturbations. In real-world applications, perturbations can come in different forms from various sources and are primarily random. However, regardless of these factors, a GNN's ability to be stable and output accurate information despite the perturbation is crucial in its applicability in the real world.

It is possible to set theoretical bounds to the impact a perturbation can have on a GNN output, which has been done so in the past[13]. It was also shown that in practice, these theoretical bounds appear to be rather loose[10]. However, the research on the looseness of these theoretical bounds uses public graph datasets, whose topologies graph properties are constrained within a small range[17]. Therefore, the exact nature of the relationship between different topological graph properties and the stability of a GNN is still unknown.

The objective of this paper is to investigate how the stability of graph convolutional networks (GCN) [11] are related to different graph properties under the semisupervised node classification problem. To achieve this, five different topological properties are analyzed using synthetically generated graphs whose selected topological properties vary within a predefined range. These graphs are then used to train a GCN, and the output for the original graph topology is computed. Finally, the deviation from the GCN's original output is measured for different random perturbations applied to the graph.

This paper is structured as follows: Section 2 summarizes other works it builds upon and provides some preliminary information on the graph properties and GNN architecture under investigation, which is necessary to understand the remaining sections. Section 3 explains in detail how the dataset generation methods work and different perturbations are introduced. Section 4 goes over the experimental setup. Section 5 presents the results and follows with a discussion of the results of the said experiment. Finally, after drawing the conclusions from the research and discussing possible future works, section 8 reflects on the paper from the perspective of responsible research.

2 Background

This section elaborates on the existing literature and preliminary information needed to understand the rest of the paper. It starts by explaining how GCNs work and continues by providing an overview of the specific implementation of GCNs being used in this research, namely topology adaptive graph convolutional networks (TAGConv) [3]. Finally, it provides definitions for different graph properties that are under investigation.

2.1 Graph Convolutional Networks

Graph convolutional networks (GCN) work very similarly to convolutional neural networks. They model data by chaining multiple layers of convolution filters and pointwise non-linear functions. However, because of their non-Euclidean structure, the regular convolution operation cannot be applied to graphs and needs to be swapped out with a more specialized version.

The convolution operation used in GCNs called *graph convolution*, makes use of the graph shift operator $\mathbf{S}$, which is an $n \times n$ matrix where n is the number of nodes in the graph, such that $\mathbf{S}_{ij}$ is zero if the nodes i and j are not connected and not zero otherwise. Using the shift operator, calculating the sum of k-hop neighbours of each node is possible via a matrix multiplication $\mathbf{S}^k\mathbf{x}$ [1], where $\mathbf{x}$ is the vector containing the value of each node. The shift operator is often the Laplacian of a graph, its adjacency matrix or their normalized counterparts [16].

Using the shift operator, the graph convolution can be defined as a simple shift and multiply operation, where $\mathbf{h}$ is the convolution filter with each element $\mathbf{h}_k$ representing the weight of the k-hop neighbors [6]. Multiple graph shift operators can be chained together into several layers with a non-

[1] For the sake of simplicity, the definitions are provided for signals with only one feature.

linear pointwise function after each layer to create a graph convolutional network. The equations (1) and (2) provide formal definitions for the graph convolution and GCNs, respectively, where $\mathbf{h}^{\ell}$ is the filter for the ℓth layer and $\sigma(\cdot)$ is the pointwise non-linear function. [2]

$$\mathbf{u}_{\ell} = \sum_{k=0}^{K} \mathbf{h}_k \mathbf{S}^k \mathbf{x}_{\ell-1} \tag{1}$$

$$\mathbf{x}_{\ell} = \sigma(\mathbf{u}_{\ell}) \tag{2}$$

2.2 Topology Adaptive GCNs

Topology adaptive graph convolutional networks (TAGConv) [3] are GNN models that leverage the irregular structure of graph data. TAGConv uses the normalized graph adjacency matrix as a shift operator. Each layer projects nodes onto the next by calculating the weighted sum of up to K^{th}-hop neighbors, approximating spectral graph convolution with K-level polynomials of the adjacency matrix. The main operation is described in (**??**), where D is the degree matrix, A the adjacency matrix, X the graph signal, and $W^{(k)}$ the weights for the k-th hop neighbors.

$$Y = \sum_{k=0}^{K} \mathbf{D}^{-\frac{1}{2}} \mathbf{A}^k \mathbf{D}^{-\frac{1}{2}} \mathbf{D} \mathbf{W}^{(k)} \tag{3}$$

Compared to other GCN implementations, such as ChebNet, TAGConv offers distinct advantages regarding flexibility and performance. It is a simple yet effective model used in state-of-the-art applications of GNNs. Typically, GCNs rely on analyzing the graph convolution spectrally and resort to approximating the graph convolution operation using Chebyshev polynomials of the graph Laplacian. To achieve a high accuracy using this method, they often need to calculate higher degree Chebyshev polynomials [1] to reach an accuracy comparable to TAGConv, or set some restrictions on the graphs that can be used with their models [11]. This leaves TAGConv as the most viable option for analyzing the stability of GNNs.

2.3 Graph Properties Under Investigation

Many properties can be calculated from a graph. This paper focuses on 6 of them. The definitions for each property under investigation are provided below, as well as an example graph in Figure 1 with values calculated for each property.

Diameter is the maximum shortest path between any pair of nodes in a graph.

Edge Connectivity is the minimum number of edges that need to be removed to make the graph disconnected.

Density is the ratio of the number of edges in the graph to the number of possible edges.

Nominal Assortativity measures the ratio of nodes of the same type being connected, rather than other ones [15]. It is calculated using the formula in equation (4), where

[2] To emphasize the graph convolution, the pooling layers have been left out of the equation.

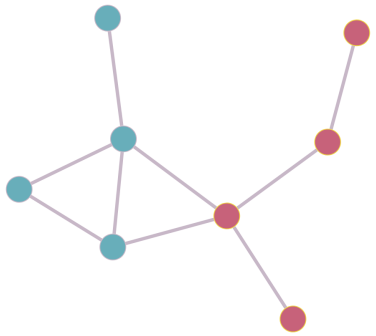


Figure 1: A graph with: 8 nodes, a diameter of 4, connectivity of 1, assortativity of 0.55 and centrality of 0.51

e_{ij} is the fraction of edges that connect nodes of type i to nodes of type j, and assortativity is A.

$$\begin{aligned} a_i &= \sum_j e_{ij} \\ b_j &= \sum_i e_{ij} \\ A &= \frac{\sum_i e_{ii} - \sum_i a_i b_i}{1 - \sum_i a_i b_i} \end{aligned} \tag{4}$$

Closeness Centrality is a metric that is inversely proportional to the average distance of a node to every other node in the graph [5]. Since closeness centrality is a metric calculated per node, this research looks into the average over all nodes.

3 Methodology

This section discusses the approach taken to measure the stability response of GNNs are trained on graphs with different properties. It begins by outlining how stability will be evaluated and the pipeline for the stability measurement process. This is followed by a description of the algorithms used to generate the synthetic graphs used in the experimental pipeline. Finally, it discusses which perturbation types are being tested for and how they are simulated.

3.1 Measuring Stability

Stability on its own is a property that cannot be measured directly, so one must resort to measuring different metrics, which can be used to make inferences on the stability of a model. One of such metrics is how much the output of a trained GNN changes when it is given a perturbed topology instead of the original one it was trained on. This difference can be measured using the relative Euclidean distance between the two outputs, which is shown in equation (5), where $g_\theta(S, x)$ is the GNN g_θ's output for a shift operator S and signal x, S is the original graph's shift operator and S_p the perturbed graph's shift operator.

$$y = \frac{\|g_\theta(\mathbf{S}, \mathbf{x}) - g_\theta(\mathbf{S}_p, \mathbf{x})\|_2}{\|g_\theta(\mathbf{S}, \mathbf{x})\|_2} \tag{5}$$

Once the assessment technique is established, the pipeline for measuring the stability of a GNN is straightforward. One crucial aspect that must be kept in mind is that it is impossible to measure a GNN's stability using only one perturbation, as many possible perturbations can exist on a graph. Instead, it is essential to test multiple different perturbations to establish an accurate representation of the GNN's stability.

3.2 Graph Generation

One challenge that must be overcome to identify a correlation between a GNN's properties and its stability is obtaining a set of graphs with high variance in many properties. However, previous literature has pointed out that existing graph datasets lack diversity [17]. Therefore, it is necessary to generate graphs to obtain meaningful results synthetically. There are many techniques to generate graphs, but in this research, only two are used: the Stochastic Block Model (SBM) and the Lancichinetti–Fortunato–Radicchi Benchmark (LFR). There is plenty of existing literature on using these models for GNN testing [17; 21; 19], and this research follows them very closely.

Stochastic Block Model

The stochastic block model (SBM) is a generative model that focuses on the creation of graphs with communities [7], making it suitable for generating synthetic graphs that will be used for node classification. It brings with it is the problem of unrealistic graphs, as they do not adhere to the power law, which is why it is a better option to use its degree-corrected counterpart [9].

To use degree-corrected SBM, it needs to be provided with four arguments: the number of nodes in the graph, n, the number of clusters in the graph, r, a $r \times r$ matrix P such that P_{ii} is the expected number of edges within the cluster i and P_{ij} where $i \neq j$ is the expected number of edges between clusters i and j. Finally, for the degree-correction, a vector θ where θ_i is the expected number of edges for vertex i must be provided.

In order to make the random sampling for these parameters more straightforward, the parameters described above will be generated from those specified below, which are all real numbers.

- **Minimum Degree** ($\theta_{min} = min_i|\theta_i|$) The minimum degree a node can have.
- **Average Degree** ($\bar{\theta}$) The average degree over all nodes.
- **p to q** (p/q) The ratio of the expected number of nodes within a cluster and in between clusters. This idea is taken from the planted partition model [4], where the diagonal and non-diagonal values of P are constant.
- **Power Exponent** (γ) The exponent used for generating a degree sequence that obeys the power law in graphs.

Given the values above, as well as the number of vertices n and number of clusters r, the hyperparameters P and θ can be calculated using the procedure in (6) and (7). An overview of how the P matrix can be derived is provided in Appendix A.

$$\mathbf{P} = \begin{pmatrix} \frac{n\bar{\theta}p/q}{r(p/q+r-1)} & \frac{n\bar{\theta}}{r(p/q+r-1)} & \cdots & \frac{n\bar{\theta}}{r(p/q+r-1)} \\ \frac{n\bar{\theta}}{r(p/q+r-1)} & \frac{n\bar{\theta}p/q}{r(p/q+r-1)} & \cdots & \frac{n\bar{\theta}}{r(p/q+r-1)} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{n\bar{\theta}}{r(p/q+r-1)} & \frac{n\bar{\theta}}{r(p/q+r-1)} & \cdots & \frac{n\bar{\theta}p/q}{r(p/q+r-1)} \end{pmatrix} \tag{6}$$

$$\begin{aligned} X &\sim U(0,1) \\ \theta_i &= ((n^{-\gamma} - \theta_{min}^{-\gamma})X + \theta_{min}^{-\gamma})^{\frac{1}{-\gamma}} \end{aligned} \tag{7}$$

Lancichinetti–Fortunato–Radicchi Benchmark

Similarly to SBM, the Lancichinetti–Fortunato–Radicchi Benchmark (LFR) also specializes in generating graphs with known communities, with the added benefit of accounting for heterogeneity in the generated graphs [12].

When the model is first created, the nodes are fully disconnected and nothing regarding the topology of the graph is known. Initially, a degree sequence is generated using the parameters k_{max}, the maximum degree in the sequence average degree $\bar{k}$ and the power law exponent τ_1. Once the degree sequence for vertices is defined, the size of clusters is then calculated according to the power law, this time using the minimum and maximum community sizes c_{min}, c_{max} and the exponent τ_2. Each node is then assigned to a community, and the graph is wired together. Once this step is complete, the entire graph is rewired without changing the node sequence such that the ratio of a node's edges to outside its community and its total degree is equal to a mixing constant $0 \leq \mu \leq 1$,

One caveat of using the LFR Benchmark is that enforcing a fixed number of classes is not straightforward. Instead, in this work, the LFR benchmark is coerced into generating a graph with r communities by providing LFR with $k_{max} = \frac{n}{r}$, $c_{min} = \frac{n}{2r}$ and $c_{max} = \frac{2n}{r}$. Since all the other parameters for LFR are just real numbers, they can easily be sampled , just like in SBM.

Overlaying Signal on Generated Graph Topologies

Once the graph topology is generated, features must be assigned to each node in the graph in a manner that is consistent with the graph's topology, meaning nodes of the same class should have reasonably similar features. In order to generate n features over a graph with r classes, the first step is to randomly select r points in the n-dimensional space, using a multivariate normal distribution with 0 mean and a covariance matrix with σ_c on the diagonal and 0 everywhere else. Call these points $\mu_1, \mu_2 ... \mu_r$. Once r points are selected, and the features for each node of any class i are randomly assigned features by sampling from a multivariate normal distribution with mean μ_i and a covariance matrix with σ_f on the diagonals. The variables σ_c and σ_f can be adjusted to control the distance between the centres of different classes and the proximity of features within the same class.

3.3 Perturbing Graphs

There are two types of topological perturbations that can occur on a graph with unweighted edges: addition and deletion

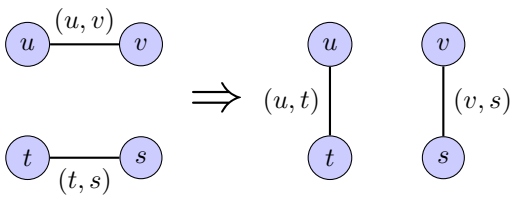


Figure 2: An illustration of the rewiring perturbation. The original edges (u, v) and (t, s) are replaced by the edges (u, t) and (v, s), maintaining the degree sequence of the graph.

of edges. This research analyzes the impact of both of those perturbations and rewiring, a particular perturbation scenario that involves both the addition and deletion of edges. When perturbations are introduced, it is always ensured that there are no parallel edges in the graph and no self-loops are created.

Addition and deletion perturbations can simply be introduced by randomly adding and deleting edges while adhering to the aforementioned criteria. However, simulating rewiring is slightly more involved. For rewiring, the objective is to add and remove edges to a graph so that the degree of each node remains the same. Because it leaves the degree sequence of the graph unchanged, it is a type of perturbation that is considerably harder to detect and thus can be used in adversarial attack [14], making it a notable perturbation to test for. In order to simulate this perturbation, four distinct edges, u, v, t, s, are selected at random such that the edges (u, v) and (t, s) exist, but (u, t) and (v, s) do not. Following this, the edges (u, v) and (t, s) are deleted and instead the edges (u, t) and (v, s) are introduced. This way, the degree sequence of the graph remains unchanged. This process is also described in figure 2.

4 Experimental Setup

This section goes over how the experiments are set up by providing detailed information on all three aspects of the experimental process: the datasets, the GNN and the perturbations. It starts by providing the hyperparameters used for generating synthetic datasets and the relevant properties of both synthetic and real-world datasets used in the experiment. It then continues by outlining the training process for TAGConv and the hyperparameters used. Finally, it goes over the testing process by detailing the amount and size of the perturbations introduced.

4.1 Datasets

In order to evaluate the relationship between graph properties and GNN stability, a dataset with a sufficient range and variety of graph properties is necessary. Therefore, synthetic datasets are generated using SBM and LFR. 2500 graphs from each model, 5000 graphs in total, are sampled for the synthetic dataset. In order to achieve variety in graph properties, the hyperparameters of each graph generation model are sampled randomly, within the ranges shown in Table 1. The two parameters that did not change between synthetic graphs are the number of classes, which is 8 and the number of features for each node, which is 16.

Parameter	**SBM**		**LFR**	
	Min	**Max**	**Min**	**Max**
# Vertices	256	1024	256	1024
Average Degree	20	40	20	40
σ_c	60	120	60	120
σ_f	4	10	4	10
Minimum Degree	5	10	N/A	
p/q	1	100	N/A	
Power Exponent	1	10	N/A	
τ_1	N/A		2	8
τ_2	N/A		1	4
μ	N/A		0	1

Table 1: Hyperparameters ranges used for graph generation. The actual hyperparameters were sampled uniformly from the provided ranges.

After testing the synthetic dataset, the results obtained are compared to real-world datasets in order to ensure that they are valid not only in a theoretical context but also in a theoretical context. For this, 3 graph datasets with different graph properties were chosen: CiteSeer, Cora and PubMed. The properties for the synthetic graph datasets, as well as the real ones, are shown in Table 2.

4.2 GNN Model and Training

In order to achieve interpretable results, the model used, as well as its hyperparameters remain the same while testing the stability of different datasets. The model used to train on the graphs is a TAGConv GNN with 4 layers, 3 of which are TAGConv layers with $K = 3$, and the final one is a linear classification layer. The first layer accepts input with n features, where n is the number of features on each node and outputs 4 features. The second layer also outputs 4 features. Finally, the last layer outputs 2 features, which are then fed to a linear classifier. In between each layer, a hyperbolic tangent $tanh$ is placed as the activation function.

The training process is also invariant across the testing of each graph. For a given graph, the nodes are split into two sets: training and testing at a ratio of 70% to 30%. The training set is populated such that 70% of the nodes from each class belong to it in order to ensure that there are samples from each node type in the training set. The testing set is then used to measure how well the model generalizes and whether it overfits. After the dataset is split, the model is trained for 300 epochs, using the Adam optimizer with a learning rate of 0.01 to optimize the cross entropy loss function for classification accuracy.

4.3 Perturbations

As explained in methodology, a graph can have different error responses to the the same type of perturbation, depending on the graph's topology and which edges the perturbation effects. Therefore, in order to obtain an accurate estimate of a GNN's stability, this research will apply the same perturbation, i.e., the same type and size multiple times, each one affecting a different random subset of edges.

In total, the three types of perturbations are tested: addition, deletion, and rewiring, and different sizes of each per-

	LFR			SBM			Citeseer	Cora	Pubmed
	Max	**Mean**	**Min**	**Max**	**Mean**	**Min**	**-**	**-**	**-**
Assortativity	0.8855	0.8715	0.8453	0.9221	0.7515	0.0029	0.6707	0.7711	0.686
Centrality	0.4724	0.388	0.3359	0.5227	0.3916	0.3223	0.3516	0.2189	0.1603
Connectivity	0	0	0	21	8.5068	0	0	0	2
Density	0.0606	0.0234	0.0096	0.1129	0.0125	0.0254	0.0016	0.0029	0.0005
Diameter	17	11.068	7	8	4.1356	3	28	19	18

Table 2: Comparison of Graph Properties Across Synthetic and Real-world Datasets. Maximum (Max), Mean, and Minimum (Min) values for assortativity, centrality, connectivity, density, and diameter metrics are presented for synthetic datasets generated using LFR and SBM models alongside real-world datasets CiteSeer, Cora, and PubMed.

turbation type. A perturbation's size is calculated relative to the graph's size, such that for addition and deletion, the edges created and removed are calculated as a percentage of the total number of edges in the graph. For rewiring, the perturbation's size is also calculated relative to the graph's number of edges, however, since rewiring is an operation that operates on two edges, the number of rewiring operations in a rewiring perturbation is calculated as a percentage of half of the existing edges in the graph. As an example, for a graph with 100 edges, a 10% addition, deletion and rewiring perturbation would involve 10 addition, 10 deletions and 5 rewiring operations, respectively.

In the final experiment, each GNN is tested for perturbation sizes 1%, 5%, 10%, and 20%, for each perturbation type. For synthetic graphs, each combination of type and size is tested with 15 different versions. Since there are many graphs in the synthetic dataset, 15 variations are sufficient to understand the impact of a perturbation type and size. However, when testing with real-world datasets, since there is only one graph in each one and some of them are relatively large, each perturbation type-size combination is tested 100 times.

5 Results

This section showcases the results of the experiments. It uses the measurements obtained from the synthetic datasets in order to identify how different graph properties impact the stability of the GNN against different perturbation types. It then discusses why the results do not indicate a notable correlation between diameter and GNN stability. Finally, it compares the results from the synthetic data with results from real-world data, discussing possible reasons why the results diverge.

5.1 Different Perturbations vs. Properties

Looking at the results obtained from the synthetic dataset, it is possible to identify relationships between graph properties and perturbation types. This section lists some of such relationships.

Assortativity

The most significant relationship that can be seen in the results is the one between assortativity and addition perturbation. As a general trend, as assortativity increases, the Euclidean distance, i.e., the error between the non-perturbed and perturbed GNN output, also increases.

Looking at Figure 3, it can be seen that both the error and its variance increase as the assortativity rises, with the rate of change in error increasing proportionally to the perturbation size.

The behaviour is unexpected from an intuitive standpoint, as the more connections that exist between nodes of the same type, the more robust the topology would be. However, this effect could be caused by the fact that as the graph becomes more assortative, the topology also carries more information about the node types. This would mean that the GNN is able to make more use of neighbouring nodes in order to classify a node, which in turn results in the model placing more emphasis on the graph's topology. This results in the model being more sensitive to perturbations, as the perturbations would end up adding unexpected erroneous information to the topology, which the model relies on but has not trained to account for. There exist training techniques that introduce addition perturbations in between different epochs in order to counteract this effect [22], and this behaviour serves to emphasize the importance of such training methods, especially on graphs with high assortativity.

Contrary to addition perturbations, highly assortative graphs are more resilient to deletions. When edges are removed, the error between the non-perturbed and perturbed GNN output decreases as assortativity rises. Robust connections within nodes of the same type help maintain graph stability, making the GNN less affected by deletions. Figure 7 showcasing the relationship of assortativity and error from deletion can be found in appendix B.

Density and Edge Connectivity

There is also a note-worthy pattern between the error caused by deletion perturbations and the density and edge connectivity of a graph. One would expect that the more edges there are in the graph, i.e., the more connected it is, the more resilient it becomes to deletions. However, this does not appear to be the case, as can be seen in Figure 4. Instead, as the density goes up, the error increases and the confidence intervals get wider. Meanwhile, higher values of edge connectivity result in an overall decrease in the error, as well as a significant decrease in the standard deviation of errors. The change in the standard deviation of errors for different in-edge connectivity values can be found in Figure **??**, in Appendix B. This behaviour makes it harder to introduce adversarial perturbations to graphs with higher edge connectivity, as it is harder to disconnect such graphs.

This behaviour can be explained by how TAGConv works, which is by approximating the graph convolution operation

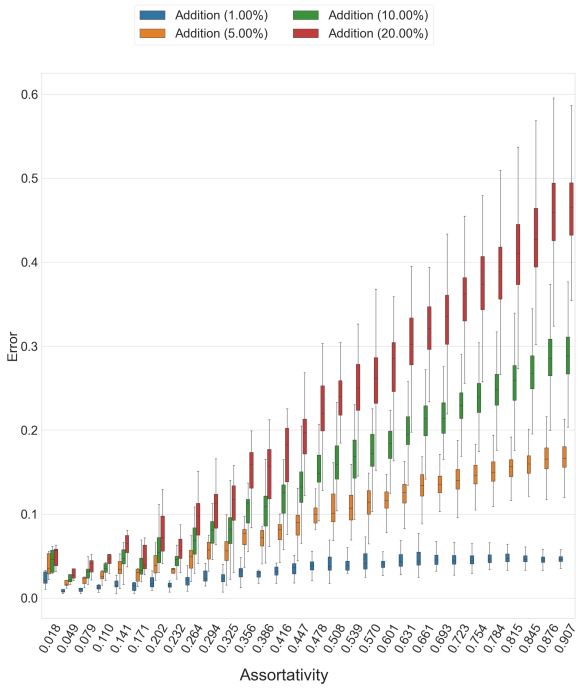


Figure 3: Box plot displaying how the average error to the addition of edges changes with assortativity, shown for different perturbation sizes.

up to the K^{th} hop neighbour of a given node. As a graph gets denser, any given node is likely to have more nodes connected to it, assuming the edges are somewhat evenly distributed. This effect grows for each hop to the next neighbour, meaning as a graph gets denser, the number of nodes affecting the prediction for a node increases significantly. However, this also makes it such that when a single edge is deleted, the amount of nodes that are being disconnected is also significantly higher. This results in less stability and higher errors as graphs get denser.

Centrality and Density

The final graph property that has a notable impact on GNN stability is the graph's average closeness centrality. As stated in section 2, closeness centrality is a metric defined inversely to the shortest distance a node has to every other node in the graph, meaning a node with a high closeness centrality is closer to every other node in the graph, and thus, is more central. Therefore, a higher closeness centrality means that most nodes are close to every other node, and the graph is more tightly connected.

The expected relationship between centrality and stability would be a positive one, i.e., stability increases with centrality. This is not the behaviour shown by the synthetic graphs. In the results obtained, which can be seen in Figure 5, the impact of centrality is highly dependent on the density of the graph as well as the size of the perturbation. For small perturbation sizes, graphs with higher centrality exhibit higher stability. Meanwhile, for larger perturbations, an increase in centrality has a detrimental effect on stability, which is amplified in graphs with high densities.

As stated in the previous section, the impact of deletion is higher for graphs with higher density. This is because for graphs with higher density, the deletion of an edge results in more nodes being disconnected from each other. For low-density graphs, when a deletion occurs, not many nodes are disconnected. Thus, a higher centrality can compensate for the few number of node disconnections, resulting in a decrease in error as centrality goes up. Meanwhile, for graphs with high densities, high centrality has a lapsing effect on stability. This could be because more nodes can be reached from any node by following a short path. As each node has a higher number of edges connected to it due to high density, centrality amplifies the impact of increasing density in graphs.

Centrality is the only property that exhibits a shift in behaviour as perturbation size increases. This relationship is similar to the one observed between density and centrality, discussed in the previous paragraph, and thus can be explained in the same manner. For small perturbations, high centrality graphs can compensate for the impact of small perturbations since the few edges being removed are not enough to disconnect enough nodes, and the disconnected nodes can still be reached via a short path. However, as the perturbation gets bigger, high centrality starts having an adverse effect on stability as a result of the same effect as the one explained for density.

After interpreting the results for closeness centrality, one thing becomes apparent: graph density, centrality, and perturbation size are all linked to the stability of the graph, and there exist thresholds which define whether an increase in centrality is detrimental or beneficial to a graph's stability.

Diameter

The final property under investigation, diameter, led to more inconclusive results compared to its counterparts. This was mainly because it was not possible to create graphs with high diameter and high density using general-purpose graph generation algorithms such as SBM and LFR. However, even though the results were not definitive, the general trend is that graphs with lower diameters appear more stable compared to graphs with higher diameters. This is presumably because graphs with low diameters are better connected and, thus, are more resilient to perturbations. Figure 10, displaying the relationship between density and diameter, can be found in Appendix B.

5.2 Impact of Rewiring

Given that rewiring perturbations are a specific form of perturbation that can be composed of addition and deletion operations, its impact on the GNN should also be similar to both addition and deletion perturbations. This is indeed the case for the synthetic graphs. Even though the impact of rewiring is strikingly similar to addition perturbations, which is likely caused by addition causing an overall larger error than deletion, it can be seen that the impact of rewiring falls in between those of deletion and addition. This can be seen in Figure 11 in Appendix B.

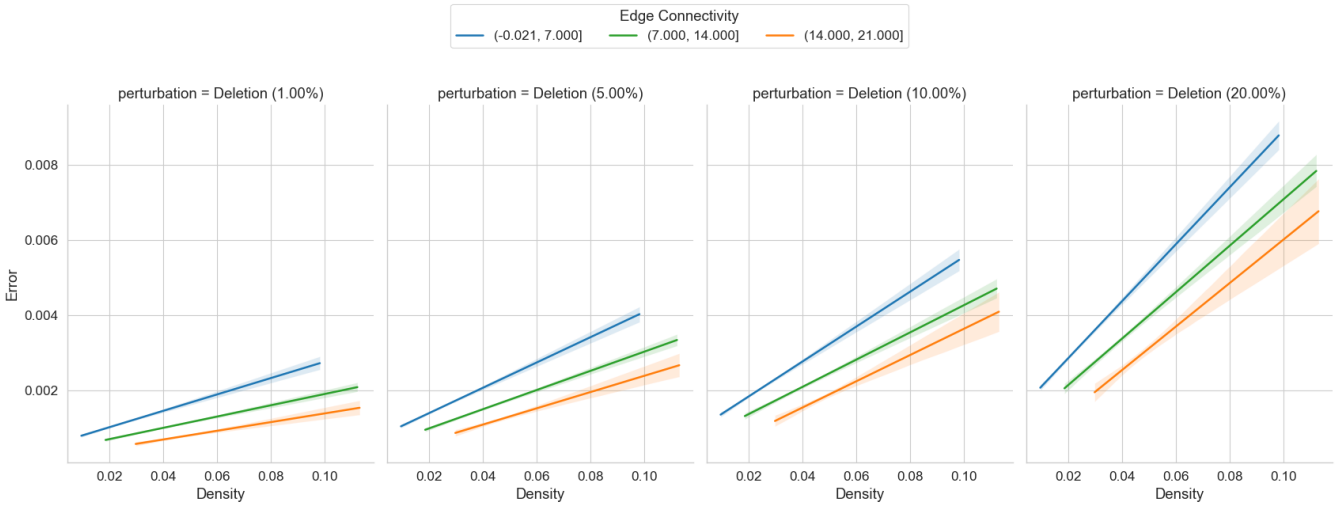


Figure 4: Plots showcasing how the mean error of each graph from deletion perturbation responds to density for graphs with different edge connectivity and perturbation sizes. The lines are fit to the data using linear regression, with the light colour hues around them representing the confidence intervals of 95%.

5.3 Real-World Dataset Comparison

After perturbing the real-world graph datasets, the errors caused by the perturbations are measured, and the mean error for each graph dataset, perturbation type and size can be found in Table 3. Unsurprisingly, the behaviour shown by the real-world datasets is different from the one observed for synthetic graphs, especially for addition perturbation. In general, it can be said that for any perturbation type, the stability is ordered such that **CiteSeer** $\geq$ **Cora** $>$**PubMed**.

Type	Size	Cora	CiteSeer	PubMed
Addition	**1.00%**	0.1493	0.1645	0.1088
	5.00%	0.3286	0.3483	0.2403
	10.00%	0.4397	0.4567	0.3154
	20.00%	0.5656	0.566	0.3965
Deletion	**1.00%**	0.0272	0.0267	0.0221
	5.00%	0.0689	0.079	0.0549
	10.00%	0.1129	0.1354	0.0871
	20.00%	0.2052	0.2553	0.1531
Rewire	**1.00%**	0.0659	0.0775	0.0405
	5.00%	0.1534	0.1732	0.0982
	10.00%	0.2251	0.2445	0.1491
	20.00%	0.3338	0.3458	0.2287

Table 3: Mean error values for each real-world graph dataset (Cora, CiteSeer, PubMed) across different types (Addition, Deletion, Rewire) and sizes (1%, 5%, 10%, 20%) of perturbations.

Based on the results from synthetic data, the expected error from the addition of the datasets would be **Cora** $>$**PubMed** $>$**CiteSeer**, since the assortativity of Cora is the greatest and PubMed has a much lower centrality compared to CiteSeer. Because the actual results are not in that order, there must be additional factors in play. From a preliminary analysis of the graph visualizations shown in Figure 8 in Appendix B, the first main difference between the graphs is the number of connected components present in each. More specifically, Cora has 78, CiteSeer has 438 and PubMed has 1 connected component in total. This difference in graph topologies can be used to explain the mismatch between synthetic and natural results. Mainly because it is sensible that addition can have a more significant impact on graphs with many connected components, as it can connect two components that were previously disconnected.

This hypothesis is further supported by the fact that Cora has lower errors for smaller perturbations, but the difference between Cora and CiteSeer decreases as perturbation sizes get larger. Looking at Figure 3, it can be seen that the impact of assortativity becomes more significant as perturbation size grows. This results in Cora's error increasing faster than CiteSeer's since it has a higher assortativity, allowing it to match CiteSeer's error.

The response to deletion is marginally more in line with the behaviour observed from the synthetic dataset. As the dataset has significantly less density, PubMed is the most robust dataset against deletion. This follows from the synthetic results. However, looking at Cora and CiteSeer, CiteSeer performs significantly worse than Cora, even though it has a lower density and higher centrality. Even though this is not necessarily a deviation from synthetic results, especially since CiteSeer has a lower assortativity, it is possible that the difference between assortativity is not the only reason behind this behaviour, and just like in addition, the more disconnected topology of the graph could hypothetically play a role in these results.

6 Conclusions

This work analyzes the relationship between various graph properties and the stability of GNNs. In particular, it mea-

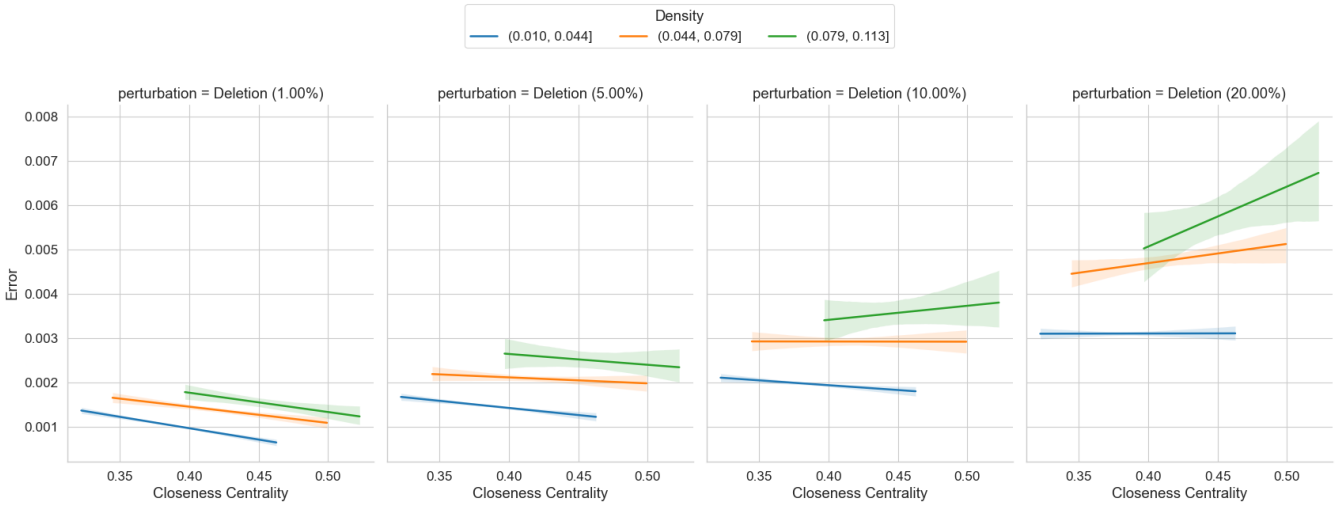


Figure 5: Plots showcasing how the mean error of each graph from deletion perturbation responds to density for graphs with different edge connectivity and perturbation sizes. The lines are fit to the data using linear regression, with the light colour hues around them representing the confidence intervals of 95%.

sures the stability of TAGConv, a GCN model, against three different types of perturbations: addition, deletion, and rewiring. This is achieved by measuring the relative Euclidean distance between the GNN output for the unperturbed and perturbed graph topologies.

To obtain a comprehensive set of results, a preliminary analysis is conducted on a synthetic dataset, which was generated using two graph generation models: SBM and LFR. The results from the synthetic dataset lead to the conclusion that as graphs become more assortative, they become more vulnerable to addition perturbations but are more resilient against deletion. Meanwhile, the opposite was the case for density and centrality. The synthetic results have also shown that the relationship between rewiring and graph properties is a combination of deletion and addition. This aligns with the fact that rewiring is just a special combination of addition and deletion perturbations.

The results from the real-world datasets were partially consistent with those obtained from the synthetic dataset. The inconsistencies that were present between the real result and synthetic results, which were anticipated, were likely caused by additional features that were not tested within the scope of this paper, such as the number of connected components in graphs. However, the conclusions that were drawn from the synthetic analysis were still mostly applicable to real-world data, especially for deletion perturbations.

Ultimately, this work is a preliminary analysis of how graph properties can impact the stability of a GNN. It can be used as a baseline for creating techniques to estimate the stability of a GNN based on several heuristics and assess its stability before being deployed in public applications.

7 Future and Related Work

While this paper serves as a preliminary analysis of the impact of different graph properties on the TAGConv GCN implementation, it also opens up numerous possibilities for future research. The landscape of graph neural networks and graph topologies presents many avenues for exploration, and this work can catalyze more in-depth studies into the stability properties of different graph topologies.

Firstly, research into different graph properties is necessary, as there are many more properties than those analyzed here. This research has shown that the number of connected components likely impacts stability against addition perturbations without explicitly testing for it. Moreover, it was impossible to generate graphs with certain properties, leaving them as a question for further analysis. For instance, it was not possible to sample graphs with high density and high diameter, leading to inconclusive results regarding the impact of diameter.

This work can also be extended to include different GNN architectures such as graph attention networks (GAT) and different implementations of GCNs such as ChebNet to uncover novel interactions between graph topology and GNN architectures. Additionally, how different graph topologies impact the stability of different GNN tasks remains an open point for research.

Of the possible avenues for further research mentioned above, preliminary research into them has already been conducted by our team. Alex Brown has studied how different perturbation types and strategies impact the stability of TAGConv GNNs, and Vladimir Rullens and Khoa Nguyen explored the stability properties of different GNN architectures for various tasks.

8 Responsible Research

This section critically reviews the rest of the paper from the perspective of responsible research. It evaluates the research from three perspectives ethical considerations, reproducibility and citations.

8.1 Ethical Considerations

The nature of this research is not one that generates many ethical questions. This is because all the data used in the experiments were either synthetically generated, or were datasets generated from public academic indexes, that are commonly used in GNN research. Therefore, the data used in the paper does not prompt for any ethical inquiries.

Moreover, the ethical considerations associated with the potential misuse of this research are mitigated by the overall benefits and advancements it brings to the field of graph neural networks. The insights gained can enhance the robustness and stability of GNN applications by highlighting potential vulnerabilities, allowing researchers and practitioners to address these weaknesses proactively. By focusing on improving the security and reliability of GNNs, the research contributes positively to the field, ensuring that the technology can be deployed more safely and effectively in various applications. This proactive approach to potential risks, coupled with the open and transparent use of public datasets, highlights the ethical soundness of the research.

8.2 Reproducibility

During the entire research process, especially in the experimentation phase, reproducibility was kept in mind. The entire experimental pipeline has been designed in a way that it can be re-executed from scratch, and the code for the entire paper is shared online publicly[3]. The repository where the code is shared comes with extensive documentation on how to run the experiments, and how to tweak the parameters for further testing.

Aside from the code, the second most important aspect of the research is the data used, and the models that are trained on this data before being tested. Even though the data can be re-generated using the code shared, since the graphs and perturbations are random, it is not guaranteed that the same results will be yielded. Therefore, the data and the models used in the experiments are also shared publicly in an online repository [4]. This repository also contains more fine-grained information, such as the Euclidean distance and cosine similarity between model outputs for each perturbation applied, to every graph.

8.3 Citations

The research makes use of several different open source libraries and tools in order to ease the development process. During the development process of the experimental pipeline, it was ensured that the licenses of the used libraries allowed for their use in academic and/or non-commercial cases, and whether citations were required. In the case that the tool's license called for a citation, such as CosmoGraph, it was cited in the bibliography accordingly.

[3] https://github.com/arg3t/GNN_stability_and_features

[4] https://huggingface.co/datasets/arg3t/GNN_stability_and_features

References

[1] Michaël Defferrard, Xavier Bresson, and Pierre Vandergheynst. Convolutional Neural Networks on Graphs with Fast Localized Spectral Filtering, February 2017. arXiv:1606.09375 [cs, stat].

[2] Xiaowen Dong, Dorina Thanou, Laura Toni, Michael M. Bronstein, and Pascal Frossard. Graph signal processing for machine learning: A review and new perspectives. *CoRR*, abs/2007.16061, 2020. arXiv: 2007.16061.

[3] Jian Du, Shanghang Zhang, Guanhang Wu, Jose M. F. Moura, and Soummya Kar. Topology Adaptive Graph Convolutional Networks, February 2018. arXiv:1710.10370 [cs, stat].

[4] Santo Fortunato. Community detection in graphs. *Physics Reports*, 486(3-5):75–174, February 2010. arXiv:0906.0612 [cond-mat, physics:physics, q-bio].

[5] Linton C. Freeman. Centrality in social networks conceptual clarification. *Social Networks*, 1(3):215–239, January 1978.

[6] Fernando Gama, Elvin Isufi, Geert Leus, and Alejandro Ribeiro. Graphs, Convolutions, and Neural Networks. *CoRR*, abs/2003.03777, 2020. arXiv: 2003.03777.

[7] Paul W. Holland, Kathryn Blackmond Laskey, and Samuel Leinhardt. Stochastic blockmodels: First steps. *Social Networks*, 5(2):109–137, June 1983.

[8] Dejun Jiang, Zhenxing Wu, Chang-Yu Hsieh, Guangyong Chen, Ben Liao, Zhe Wang, Chao Shen, Dongsheng Cao, Jian Wu, and Tingjun Hou. Could graph neural networks learn better molecular representation for drug discovery? A comparison study of descriptor-based and graph-based models. *Journal of Cheminformatics*, 13(1), February 2021. Publisher: Springer Science and Business Media LLC.

[9] Brian Karrer and M. E. J. Newman. Stochastic blockmodels and community structure in networks. *Phys. Rev. E*, 83(1):016107, January 2011. Publisher: American Physical Society.

[10] Henry Kenlay, Dorina Thanou, and Xiaowen Dong. Interpretable Stability Bounds for Spectral Graph Filters, February 2021. arXiv:2102.09587 [cs].

[11] Thomas N. Kipf and Max Welling. Semi-Supervised Classification with Graph Convolutional Networks, February 2017. arXiv:1609.02907 [cs, stat].

[12] Andrea Lancichinetti, Santo Fortunato, and Filippo Radicchi. Benchmark graphs for testing community detection algorithms. *Physical Review E*, 78(4):046110, October 2008. arXiv:0805.4770 [physics].

[13] Ron Levie, Elvin Isufi, and Gitta Kutyniok. On the Transferability of Spectral Graph Filters. *CoRR*, abs/1901.10524, 2019. arXiv: 1901.10524.

[14] Yao Ma, Suhang Wang, Lingfei Wu, and Jiliang Tang. Attacking Graph Convolutional Networks via Rewiring. *CoRR*, abs/1906.03750, 2019. arXiv: 1906.03750.

[15] M. E. J. Newman. Mixing patterns in networks. *Physical Review E*, 67(2):026126, February 2003. arXiv:cond-mat/0209450.

[16] Antonio Ortega, Pascal Frossard, Jelena Kovačević, José M. F. Moura, and Pierre Vandergheynst. Graph Signal Processing: Overview, Challenges, and Applications. *Proceedings of the IEEE*, 106(5):808–828, May 2018.

[17] John Palowitch, Anton Tsitsulin, Brandon Mayer, and Bryan Perozzi. GraphWorld: Fake Graphs Bring Real Insights for GNNs. In *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, KDD '22. ACM, August 2022.

[18] N. Rokotyan, O. Stukova, D. Kolmakova, and D. Ovsyannikov. Cosmograph: GPU-accelerated Force Graph Layout and Rendering, 2022.

[19] Neil Shah. Scale-Free, Attributed and Class-Assortative Graph Generation to Facilitate Introspection of Graph Neural Networks. *San Diego*, 2020.

[20] Namita Shrivastava, Amit Bhagat, and Rajit Nair. Graph Powered Machine Learning in Smart Sensor Networks. pages 209–226. January 2022.

[21] Mustafa Yasir, John Palowitch, Anton Tsitsulin, Long Tran-Thanh, and Bryan Perozzi. Examining the Effects of Degree Distribution and Homophily in Graph Learning Models, July 2023. arXiv:2307.08881 [cs].

[22] Tong Zhao, Yozen Liu, Leonardo Neves, Oliver Woodford, Meng Jiang, and Neil Shah. Data Augmentation for Graph Neural Networks, December 2020. arXiv:2006.06830 [cs, stat].

9 Appendix

A Deriving the P Matrix for Stochastic Block Model

As a requirement of the library used to generate the SBM graphs[5], a P matrix that contains the average number of expected edges in between each class is needed. This section attempts to derive this matrix by constructing it for a small example with 3 classes and generalizing it afterward.

Take a graph with 3 classes, $r = 3$, with n nodes and a target mean degree of $\bar{\theta}$, with p/q as the ratio of intra-class edges to inter-class edges. A naive first approach to take in order to ensure the final graph has a mean degree of $\bar{\theta}$ and class edge ratio of p/q is drawing a graph with $\bar{\theta}$ edges in between each cluster $p\bar{\theta}$ edges within a cluster, such a graph can be found in Figure 6. Notice that the edges are directed in the graph shown, as the P matrix applies to directed graphs, the only difference is that it must be symmetric for undirected graphs.

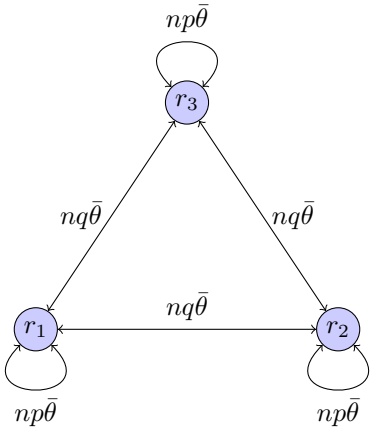


Figure 6: A sample graph with 3 different clusters, visualized. Each node represents a cluster of nodes, and the edges represent the expected number of edges between the clusters. The mean degree of the graph is, $\bar{\theta}$ and the ratio of intra-class to inter-class edges is p/q.

However, calculating the average degree over all nodes for this first iteration of the graph, it can be seen the mean is not in fact $\bar{\theta}$ but $\bar{\theta}3(p + 2q)$. Therefore, the number of edges must be corrected by a factor of $\frac{1}{3(p+2q)}$ in order to achieve a mean degree of $\bar{\theta}$.

In the $r = 3$ example, the constants 3 and 2 were introduced into the equation due to the number of classes. For a graph with an arbitrary number of clusters r, connected together using the aforementioned naive strategy, the factor of correction would be $(r(p + (r - 1)q))^{-1}$. This is because an arbitrary class r_x would have $n(r - 1)q\bar{\theta}$ edges leaving from it, and coming from other clusters, as well as $np\bar{\theta}$ edges within itself. This means in total there are $n\bar{\theta}r(p + (r - 1)q)$ edges and in order to achieve a mean number of edges of $\bar{\theta}$, the number of edges should be corrected by the factor $(r(p + (r - 1)q))^{-1}$.

As a final step, in order to have the expected number of edges on each entry of the matrix, the matrix is multiplied with n, the number of edges. And in order to simplify it, p is set to p/q and $q = 1$. These final touch-ups result in the matrix shown in Section 3.

[5] graph-tool: https://graph-tool.skewed.de/

B Additional Figures and Results

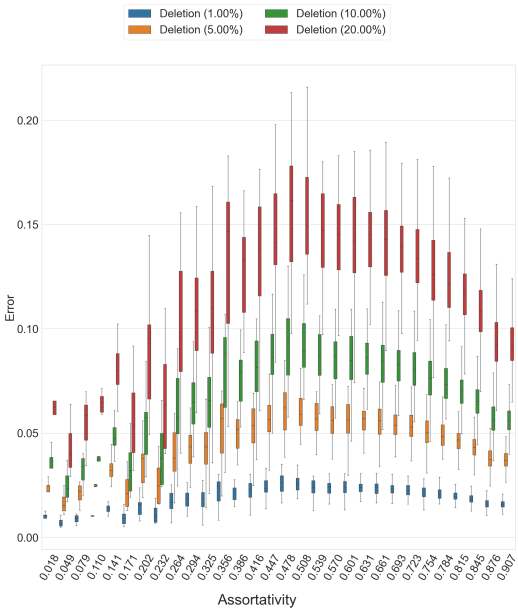
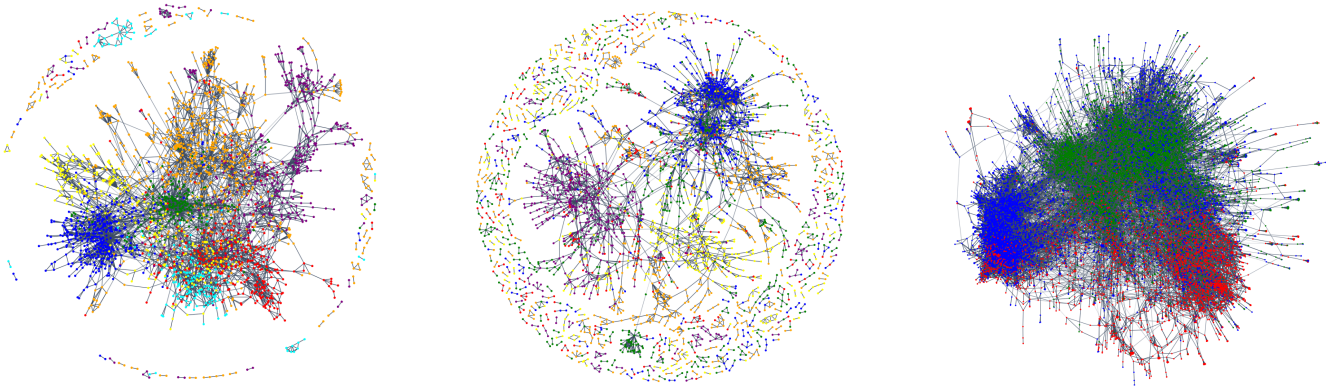


Figure 7: Box plot displaying how the average error to deletion of edges changes with assortativity, shown for different perturbation sizes.



(a) Cora (b) CiteSeer (c) PubMed

Figure 8: The 3 real world graph datasets under investigation, visualized using CosmoGraph[18]. A node's color represents its class.

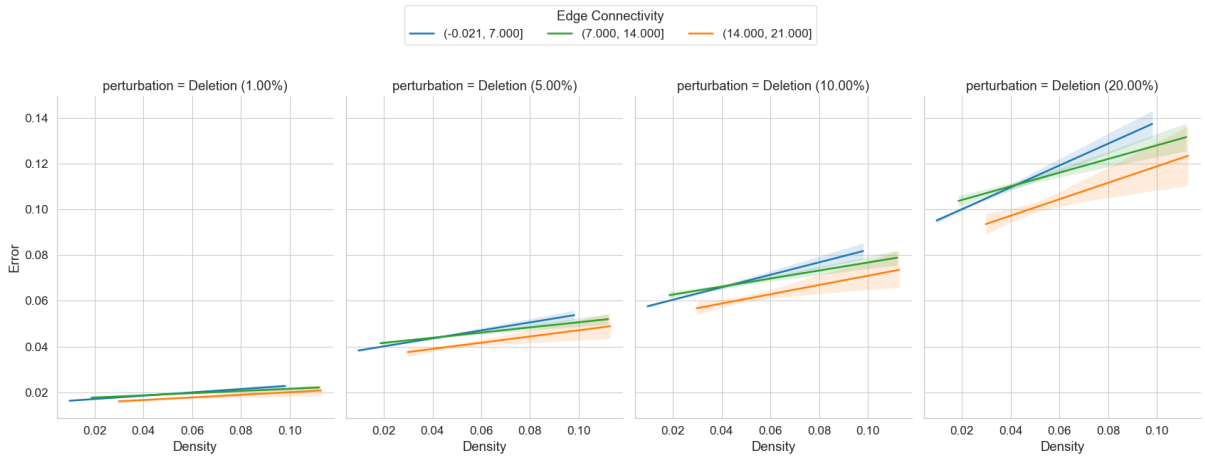


Figure 9: Plots showcasing how the variance of the error from deletion perturbation responds to density, for graphs with different edge connectivity and perturbation sizes. The lines are fit to the data using linear regression, with the light color hues around them representing the confidence intervals of 95%.

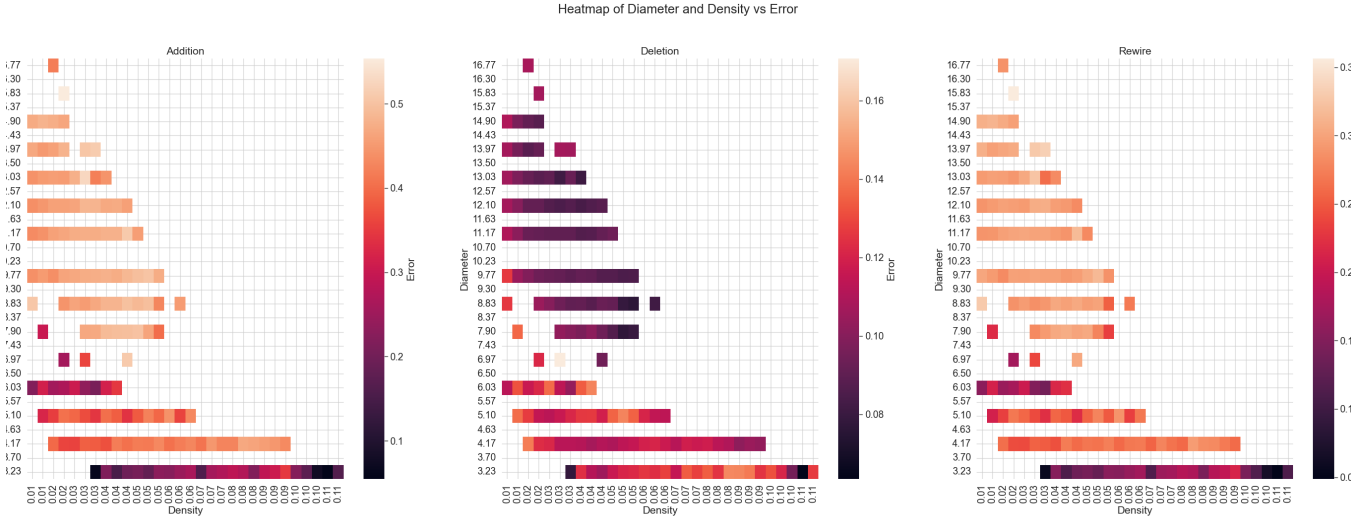


Figure 10: Heatmap showing how the average error changes with density and diameter, shown for different perturbation types. It also highlights the absence of graphs with high density and diameter in the synthetic dataset.

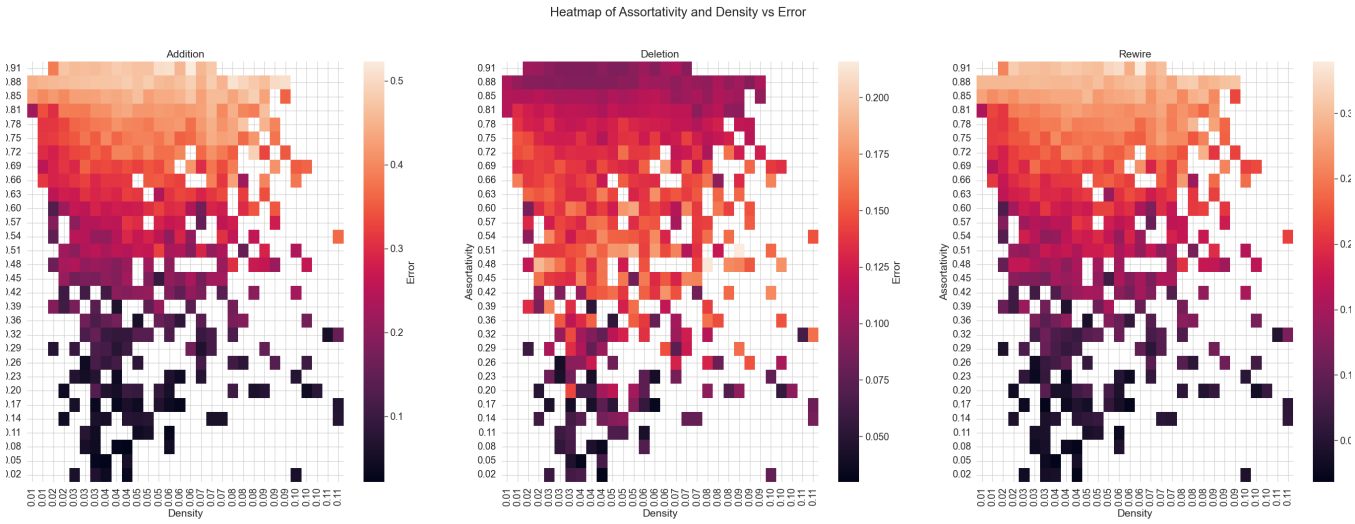


Figure 11: Heatmap showing how the average error changes with density and assortativity, shown for different perturbation types. It can be seen that the impact of rewiring is very similar to addition but with less error. This is caused by the fact that rewiring is a combination of addition and deletion.