5-14-2023

Graph Augmentation using Spectral moments

Atul Anand Gopalakrishnan

Syracuse University

Abstract

Graph representational learning focuses on learning real value vectors that for nodes,edges or the graph, such that these vectors capture adequate information about these entities. Graph data augmentation, focuses on changing the structure or features in a graph to help improve classification performance and become more generalizable. This can be broadly categorized into feature based augmentation and structure based augmentation. Feature augmentation focuses on changing the feature matrix, without changing the structure of the graph to help improve the performance of the graph neural network. Graph structure augmentation refers to the manipulation of the adjacency matrix of a given graph to achieve better classification performance.

Our approach focuses on the problem of graph augmentation but from a spectral standpoint. More specifically, we attempt to augment a graph using spectral moments. Recent results have indicated that the second, third and fourth spectral moments of a graph, have strong connections to the graph's properties, such as degree distribution, clustering coefficient, and connectivity [1]. Our contribution is two fold: First, we explain a formal method to find a spectral moment that helps maximize node classification performance. Second, we also provide an algorithm to augment the graph using it's spectral moments, and therefore augment the graph to the spectral point that helps maximize classification performance while making the graph sparse. For the purpose of node classification, we use the GraphSAGE model with no node sampling and the mean aggregator. We notice that the node classification performance after augmentation goes up in a majority of our datasets, and furthermore, the graph also gets sparser across all our datasets.

Graph Augmentation using Spectral moments

by

Atul Anand Gopalakrishnan

B.Tech., PES University, 2021

Thesis

Submitted in partial fulfillment of the requirements for the degree of

of Master of Science in Computer Science

Syracuse University

May 2023

Acknowledgement

Firstly, I would like to express my sincere gratitude to my guide, Dr. Reza Zafarani, for his unwavering support and guidance throughout my thesis. When I first started working with him, I was a novice in the field, but his patience, expertise, and encouragement helped me to grow and succeed. It was an absolute pleasure to collaborate with him, and I am grateful for this opportunity to have learned from this experience.

Secondly, I would like to extend my sincerest gratitude to my parents, Anand Gopalakrishnan and Rajalekshmy Ramaswamy, who have been the best supporters throughout my academic journey. Their unwavering love, encouragement, and sacrifice have made this achievement possible. Their constant belief in me has been a great source of motivation, and I am incredibly grateful for their guidance and support. This thesis is a testament to their love and dedication, and I could not have done it without them.

Finally, I would like to thank my thesis committee members, Dr. Edmund Yu, Dr. Sucheta Sounderajan, and Dr. Garrett Katz. Without their participation and valuable input, the defense could not have been successfully conducted.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

Graphs have always found a place at the forefront of computation. In today's world, much work has gone into using graphs for machine learning-based tasks applied in domains such as social networks [2], biology [3], and chemistry [4]. Graph-based machine learning methods have also benefited from other domains like natural language processing [5] and computer vision [6]. As a part of this process, one problem we deal with is representational learning in graphs. *Graph Representational learning* focuses on learning embeddings for nodes, edges, or the graph itself to capture some information using the graph structure adequately.

The earliest method used in graph representational learning is graph signal processing(GSP). The motive is to learn a mapping function f that maps nodes to real value vectors known as vectors which are known as *signals*. We can define GSP in either the spatial domain or the spectral domain. The spatial domain tries to ensure the signals learned are *smooth*, which states that the sum of the square difference in the signals of a node and its neighbors should be minimal. We use the Graph Fourier Transform to learn graph signals in the spectral domain. When we apply Fourier transforms to a graph, we learn coefficients that help indicate how important each component in the signal is. We filtered these coefficients to get the essential ones and reverse them into signals using the Inverse Graph Fourier Transform.

The current work in this domain also uses semi-supervised learning in graphs, with applications of spatial and spectral properties. The models used here differ from other neural network models because graphs tend to have a non-Euclidian structure associated with them, unlike images or text. The underlying concept here focuses on *Neural Message Passing*. We define Message-Passing as the process of nodes sharing messages in the form of embeddings with one another to help learn and update their embeddings. *Neural Message Passing* consists of two steps, AGGREGATE and UPDATE, where AGGREGATE collates the embeddings from a node's neighborhood, and update uses these embeddings to learn and update the node's embeddings.

The initial aggregate and update step provides a node with information from its 1-hop neighbors. We repeat this operation k times to obtain node embedding containing information about the k-hop neighborhood. Some popular methods used here include Graph Neural Networks(GNN) [7], Graph Convolutional Networks (GCN) [8], Graph Attention Networks(GAT) [9], GraphSAGE [10] and Graph Isomorphic Networks. By leveraging spectral graph theory properties, GCN derives a convolution operation that enables spatial description and accomplishes this through a dual-degree normalized method of message propagation.

GraphSAGE uses both an aggregate and an update step, concatenating the node embedding and its aggregated neighboring messages. GraphSAGE is also inductive and more generalizable to unseen nodes. Different aggregator functions in GraphSAGE include Mean Aggregator, LSTM Aggregator, and Pooling Aggregator. GAT uses attention-based weights for each edge to learn edge importance and attention coefficients. The attention coefficients are multiplied with the messages passed along each edge before adding them to update the node embeddings. Graph Isomorphism Network (GIN) passes the aggregated output to a Multi-Layer Perceptron(MLP), which applies a learnable function on the summation. GIN also measures the importance of the self-node with a parameter epsilon.

Much work has occurred in graph data augmentation in the past few years. In other machine learning domains, data augmentation is a commonly used technique for solving problems related to increasing the amount of training data or improving generalization to noisy data. However, when it comes to graphs, these augmentation strategies are only partially transferable to graphs due to their non-Euclidean nature. Graph data augmentation is motivated by the fact that networks can become misaligned from their true form. Sometimes graphs can also be noisy, for example, having inauthentic connections or not connecting with individuals you know on a social platform [11]. For this reason, augmenting graphs to achieve a more accurate alignment can be helpful. There are two methods in graph data augmentation which include *feature augmentation* and *structure augmentation*.

Feature augmentation alters/augments the features of a graph and not its structure. It involves modifying the feature matrix of the graph or the hidden representations generated by the model. Methods include *feature noising* [12] [13] that corrupts node/intermediary features during the training process to improve generalizability and reduce problems like overfitting and sensitivity to outliers. *Feature masking* [14] sets random feature vectors to 0 at each step. *Feature recovery* aims to recover corrupted features in the feature matrix, as described in papers such as citewang2020nodeaug [15], that propose methods for achieving this goal, by modifying heuristic functions that utilize the features of neighboring nodes or gradient flow.

Lastly, *feature recovery* aims to recover corrupted features in the feature matrix. One can achieve this by defining heuristic functions that use the features of neighboring nodes or gradient flow. Applying feature augmentation techniques can improve the accuracy of graph neural networks without changing the underlying graph structure [16].

Graph structure augmentation refers to the manipulation of the adjacency matrix of a given graph to achieve better classification performance. One can use these methods to carry out structural augmentation in graphs. *Edge perturbation* involves dropping edges in a graph, either randomly [17] using a corruption matrix. *Graph rewiring* modifies the graph by rewiring edges, often to counter homophily and over-quashing [18] [19]. *Graph* *sampling* [20] [21] creates subgraphs by passing a given graph through a sampler, with various types, such as vertex-based and edge-based samplers. *Node dropping and node insertion* [22] focus on dropping or adding nodes and associated edges to the graph.

We address this problem of graph augmentation from the spectral standpoint. For this, we use spectral moments, and to our understanding, we are the first to do the same. We chose spectral methods because they are shown to have strong connections to graph properties, such as degree distribution, clustering coefficient, and connectivity [1]. As a part of our method, we attempt to take a large graph's spectral moment to the best-performing subgraph. Our method, broadly stated, helps sparsify the graph and improve its classification performance. To identify the spectral moment of the subgraph with the highest node classification performance, we interpolate a surface plot of second(m_2), third(m_3), and fourth(m_4) spectral points, colorize the surface with the Micro-F1 score and Macro-F1 score, and identify a spectral moment that maximizes both of them. We have also developed an algorithm that helps augment a graph to a desired spectral moment. We use the concept of the l^{th} spectral moment(m_l), which is directly proportional to the average number of l-walks per node in the graph and the average return probability of a closed l-walks in the graph.

Chapter 2 reviews related methodologies to graph embedding and augmentation methods, highlighting their strengths and limitations. In Chapter 3, we present our methodology in detail. We outline the steps to generate surface plots and describe our algorithm to modify spectral points, which enables us to enhance the quality of the graph embeddings. Chapter 4 focuses on the data and experimental setup used in our study. We describe the datasets we used and provide an overview of the experimental design. Additionally, we present a detailed analysis of the results obtained, comparing our approach to existing methods in the literature. We also discuss possible future research directions that could build upon our findings and extend our approach. Finally, in Chapter 5, we conclude our work by summarizing the main contributions of our methodology.

Chapter 2

Related Work

In this section, we talk about the recent advent of graph embedding and augmentation models. Section 2.1 briefly introduces us to graph embedding models and further elucidates by talking about methods such as Matrix Factorization in 2.1.1, Co-occurrence based approaches in 2.1.2 and Neural Network Models in 2.1.3. Section2.2 talks about graph augmentation methods and addresses some recent work in areas such as Feature Augmentation in 2.2.1 and Structural Augmentation in 2.2.2.

2.1 Embedding Models in graphs

One of the central problems we deal with in machine learning with graphs is to develop representations that capture the graph structure information in terms of a feature vector.

2.1.1 Matrix factorization based methods

Matrix factorization focuses on regenerating the *graph's* adjacency matrix from the node representations. Matrix factorization used the encoder-decoder model to help produce embeddings. The encoder model helps produce embeddings. In the case of graphs, the encoder acts as a function that helps produce an embedding for a given node. In graph-based machine learning models, the decoder model reconstructs some graph metrics using the embeddings generated by the encoder. One can optimize these models by minimizing the reconstruction loss, which measures how accurately the decoder model captures the generated graph embeddings. The objective is to ensure that the node representations can capture information about the given similarity matrix, which provides information on pairwise node similarity, for example, *neighborhood overlap*. There are two dominant methods in this area: 1) Laplacian Eigen maps and 2) the Inner product method.

1. **Laplacian Eigenmaps** [23] use ℓ_2-Norm to capture the distance between nodes u and v and compare this distance to the corresponding value in the similarity matrix(S), which is $S_{u,v}$ here. Equation 2.1 describes the reconstruction loss used for Laplacian Eigenmaps. Here, x_u and x_v refer to the embeddings for nodes u and v, DEC is used to represent the decoder, L represents the reconstruction loss, and S represents the similarity matrix.

$$\begin{gathered}\text{DEC}(x_u, x_v) = \|\mathbf{x_u} - \mathbf{x_v}\|_2^2 \\ L = \sum_{u,v \in V} \text{DEC}(x_u, x_v) S[u, v]\end{gathered} \tag{2.1}$$

   Suppose we set the similarity matrix as the Laplacian matrix and generate embeddings that are d-dimensional. The function in Equation 2.1 penalizes the embeddings when the similarity between the features is low, but the value in the similarity matrix is high. The solution that minimizes Equation 2.1 corresponds to the d-smallest eigenvectors of the Laplacian.

2. **Inner Product Method** focuses on replacing the ℓ_2-Norm with the inner product of the graph embeddings as shown in Equation 2.2 to check if the inner product can capture the similarity value in S. The variables in Equation 2.2 are the same as those in Equation 2.1

$$\begin{gathered}\text{DEC}(x_u, x_v) = \mathbf{x_u^T x_v} \\ L = \sum_{u,v \in V} \left\|\text{DEC}(x_u, x_v) - S[u, v]\right\|_2^2\end{gathered} \tag{2.2}$$

Some methods use this approach but differ in the way they define the similarity matrix([24], [25], [26]). *Graph Factorization* [24] uses the adjacency matrix as S, thereby using the embeddings to capture connections between nodes. *GrapRep* [25] additionally uses higher powers of the adjacency matrix to capture k-step similarity. The algorithm concatenates the node embeddings for different values of k to give one single representation. The *HOPE* algorithm [26] sets S to be the *neighborhood similarity measure* between node pairs which can be measured using methods like Jaccard similarity to measure the number of common neighbors between two nodes or Katz index, which measures the number of paths between two nodes.

2.1.2 Co-occurrence based approaches

Co-occurrence is an embedding similarity measure that states that nodes co-occurring together in a walk should have high embedding similarity. Loosely there are three major approaches here which include, DEEPWALK [27], NODE2VEC [28], and LINE [29].

We commonly observe that the co-occurrence of nodes in random walks follows the Power-Law distribution model. Words in Natural Language Processing tend to follow a similar distribution. As a result, it is possible to remodel methodologies in NLP to model word frequencies like SkipGram [30] into graph representational learning. The SkipGram model tries to preserve information about sentences by taking the co-occurrence of words in that sentence into account: The goal is to maximize the co-occurrence probability among the words that appear within a window, w. We can do the same by minimizing the following objective function for graph $G = (V, E)$:

$$\min_{\Phi} \left(-\log Pr(\{v_{i-w}, \ldots, v_{i-1}, v_{i+1}, \ldots, v_{i+w}\} | \Phi(v_i)) \right) \tag{2.3}$$

Here in Equation 2.3, $v_{i-w}, \ldots v_{i-1}, v_{i+1}, \ldots, v_{i+w}$ represents the nodes in a window w

belonging to a random walk, where $v_{i-w}, \ldots v_{i-1}, v_{i+1}, \ldots, v_{i+w} \in V$ and Φ is a representation function, that can map a node v_i into a representational space, such that $\Phi(v_i) \in \mathbb{R}^d$, where d is the size of node embedding.

DEEPWALK samples a random root node from graph G and uses it as the start node for the random walk. The random walk samples a neighbor of the last visited node until we reach the desired length. The sampled nodes are passed to SkipGram, maximizing the probability of the co-located nodes within a window w.

NODE2VEC is very similar to DEEPWALK, except for its walking strategy. Typically there are two properties that NODE2VEC aims to capture: *homophily* and *structural similarity*. Homophily is when two or more connected nodes share similar properties. Structural similarity states that nodes with similar structures have similar embeddings. While homophily requires connecting similar nodes, structural similarity can be between disconnected nodes. To solve this problem, NODE2VEC shifts between two extreme walking strategies of Depth First Search(DFS) and Breadth First Search(BFS), respectively. DFS helps by providing a more macroscopic view of the graph, thereby helping out with homophily, and BFS gives an objective view of the connected components aiding with structural similarity. NODE2VEC does this by learning a search bias term α. This is defined as follows in Equation 2.4:

$$\alpha_{p,q}(t,x) = \begin{cases} \frac{1}{p} & \text{if } d_{t,x} = 0, \\ 1 & \text{if } d_{t,x} = 1, \\ \frac{1}{q} & \text{if } d_{t,x} = 2, \end{cases} \tag{2.4}$$

The parameter p is the *return parameter*. It decides the likelihood of revisiting the same node. If the value of p is high, then we explore other nodes; otherwise, we revisit the same node. The parameter q is the *inward-outward parameter*. A high value of q makes the walk more biased towards nodes closer to the current node. A low value would make the walk more biased towards further nodes from the current node.

Finally, LINE is not explicitly a random walk-based method; its goal is to preserve both first- and second-order proximity. First-order proximity ensures that nodes that are directly connected should have similar embeddings. Second-order proximity ensures that the embedding proximity between two pairs of connected nodes is proportionate to their structural similarity. We treat each node's neighbor as its context and expect two nodes with similar contexts to have similar embeddings.

Several other embedding methods use on random walks and co-occurrence. For example, WALKLETS [31] is a method similar to Deepwalk that includes skip connections over certain nodes. STRUC2VEC [32] aims to convert structural properties to representations. Their methodology does this by constructing a graph that is connected based on degree similarity and running Deepwalk on this graph to produce embeddings that preserve structural properties

2.1.3 Neural Network-based approaches

The neural network approach focuses on neural message passing which can be understood using Equation 2.5, where node $u's$ neighbors indicated as $N(v)$ pass their aggregated embedding to u. Node u then uses the aggregated embeddings to update itself. This process is repeated for all nodes in the given graph. The aggregate and update functions are explained in greater detail below.

$$h_u^{k+1} = \text{UPDATE}^k(h_u^k, \text{AGGREGATE}^k(\{h_v^k, \forall v \in N(u)\})) \tag{2.5}$$

The AGGREGATE function uses the embeddings of a node's neighbors to generate the final embedding. This is repeated over several iterations with the intuition that the initial

AGGREGATE and UPDATE functions provide a node with all information from its 1-hop neighbors. After each iteration, the node's embedding will contain information about the k-hop neighborhood. A common modification would be adding a self-loop and aggregate rather than updating.

GCNs normalize the messages using a diagonal node degree matrix to ensure that node degrees do not disrupt the scale of feature vectors. Spectral graph theory motivates symmetric normalization, and we can describe the Graph Convolutional Network using Equation 2.6.

Here,

1. $\hat{\mathbf{A}} = \mathbf{A} + \mathbf{I}$, where A is the adjacency matrix represents the *adjacency matrix*. I is added to the A to add self-loops to add the feature vector of the self-node to the aggregated sum of neighboring nodes.

2. $\hat{\mathbf{D}}$ represents the diagonal-degree matrix of $\hat{\mathbf{A}}$, where D_{ii} is the degree of node i.

3. $\mathbf{H}^{(l)}$ represents the embeddings of the nodes at layer l and $\mathbf{W}^{(l)}$ represents the weights for each layer.

Here the embeddings are aggregated by $\hat{\mathbf{A}}$ and degree normalized by $\hat{\mathbf{D}}^{-1}$ which can be decomposed as $\hat{\mathbf{D}}^{\frac{-1}{2}}\hat{\mathbf{D}}^{\frac{-1}{2}}$. Put together, this can be written as $\hat{\mathbf{D}}^{-\frac{1}{2}}\hat{\mathbf{A}}\hat{\mathbf{D}}^{-\frac{1}{2}}$

$$\mathbf{H}^{(l+1)} = \sigma(\hat{\mathbf{D}}^{-\frac{1}{2}}\hat{\mathbf{A}}\hat{\mathbf{D}}^{-\frac{1}{2}}\mathbf{H}^{(l)}\mathbf{W}^{(l)}) \tag{2.6}$$

GraphSAGE [10] uses both an aggregate and an update which involves concatenating the node embedding and its aggregated neighboring messages. GraphSAGE is also inductively compared to GCN and is more generalizable to unseen nodes. Multiple aggregator functions can be used, including Mean Aggregator, LSTM [33] aggregator, and pooling aggregator. The mean aggregator is similar to GCN's mean update and aggregation. However, unlike GCN, which performs mean aggregation on the node and its neighbors, we find the mean of neighbors'$(N(v))$ representations here in the aggregate step. Thus GraphSAGE's mean aggregator can act as an inductive variant of the GCN aggregator. The LSTM aggregator has a larger expressive capability and is commonly used in NLP. However, it is not permutation invariant as it takes inputs and processes them sequentially. GraphSAGE makes it unordered/permutation invariant by making it work on random permutations of nodes' neighbors. Max pooling is another variant of the aggregate function, where it takes the element-wise max on each feature across a node's neighborhood to obtain a node's representation.

Attention mechanisms are specifically useful when we want to focus on parts of a input that are more relevant. *Graph Attention Networks*(GAT) [9], add an attention-based weight to each edge, which is multiplied by the message passed along that edge before adding them up. GAT finds the attention weights as shown in Equation 2.7: First, we find the edge importance as shown in the equation by passing $\mathbf{W}h_i$ and $\mathbf{W}h_j$ through an attention layer. Here, the value of e_{ij} indicates the importance of node i's features to node j. Next, we normalize these coefficients using the softmax function, which is indicated as α_{ij}. Finally, the attention weights are used to add the neighbors' embeddings to give the node representation, thereby enabling us to focus on neighboring node representations that are more important.

$$\begin{aligned} e_{ij} &= a(\mathbf{W}h_i, \mathbf{W}h_j) \\ \alpha_{ij}^{(l)} &= \frac{exp(e_{ij})}{\sum_{k\in\mathcal{N}i} \exp(eik)} \\ h_i^{(l)} &= \sigma(\sum_{j\in\mathcal{N}i} \alpha\mathbf{W}h_j^{(l-1)}) \end{aligned} \tag{2.7}$$

Another robust network architecture is the *Graph Isomorphism Network*(GIN) [34]. GINs show that graph neural networks are the most representative of the *Weisfeiler-Lehman* (WL) [35] test. Briefly, isomorphism tries to check if two graphs are topologically the same. There is no known solution to check if two graphs are isomorphic, but the WL test is one of the closest to doing so. Analogous to message passing, the method aggregates 1-dimensional node labels and hashes them into new labels. After several iterations, two graphs are considered to be isomorphic if their node labels match across all nodes. The GIN architecture passes the aggregated output to a Multi-Layer Perceptron(MLP), which applies a learnable function on the summation. It also measures the importance of the *node* with a parameter ϵ, where a high value of ϵ gives more importance to the *node* in comparison to the *node*'s neighbors.

2.2 Augmentation methods

The previous section discusses various graph embedding methods that help generate node representations. As a part of this section, we talk about graph augmentation methods. Observed network data may contain errors or missing information due to incomplete data collection, noise, or sampling bias. Graph data augmentation techniques aim to address these issues by generating new graph instances with some variations that help boost the performance of existing representational models. In the following sections, we discuss two popular augmentation methods, namely *feature augmentation* and *structure augmentation*.

2.2.1 Feature Augmentation

A graph G, in the context of graph neural networks, can be defined as $G = (A, X)$, where A is the adjacency matrix and X is the feature matrix. *Feature augmentation* focuses on changing the feature matrix without changing the graph's structure to help improve the performance of the graph neural network. Feature augmentation can also work towards augmenting the hidden representations generated by the graph neural network. The following methods are used for feature augmentation in graphs:

1. **Feature noising** corrupts node features or their intermediary features in the training process. Its goal is to help improve the given model's generalizability, reduce overfitting, and make graph embedding models more stable to outliers. This noise can either be initialized arbitrarily [12] or trained using adverserial [13] graph of the given graph.

2. **Feature masking** hides some of the features propagated at each step. The idea is similar to corrupting, except some feature vectors are set to zero. Often, we can observe that setting the features arbitrarily to zero changes the distribution of the given graph feature vectors. To solve this, most methods that fall in this category sample from a given distribution that matches the graph properties [14] [36].

3. **Feature recovery** focuses on recovering corrupt features that are nosy or incomplete. Recovering corrupt features can help improve the accuracy of the given graph embedding model. This is done rewriting each vector(x_i) with another vector that can be learnt(b_i) as mentioned in Equation 2.8:

$$\mathrm{x}_i^{new} = \alpha \mathrm{x}_i + \beta \mathrm{b}_i \tag{2.8}$$

   where α_i and β_i are controlling parameters. The methods using feature recovery differ in how they define b_i. [37] uses neighboring features and [15] uses gradient flow to learn b_i.

4. **Feature combining** focuses on generating new features based on the current set of features X. Here, two or more features are combined using a weighted method to produce a new synthetic feature. This can be done on the intermediate results as well [38].

2.2.2 Graph structure Augmentation

Graph structure augmentation is the process of changing the adjacency matrix of a given graph to provide an optimal improvement in classification performance. There are a couple of methods to perform structural augmentation in graphs.

1. **Edge Perturbation** is done by dropping edges in a graph. The simplest way to do this is to drop random edges, like DropEdge [17]. Mathematically edge perturbation drops edges using a corruption matrix(C). C has the same dimensionality as the adjacency matrix and determines the edges to drop by setting the respective edges to 1. The values of C are usually sampled from a distribution, where edges are set to 1 or 0 based on some probability.

2. **Graph Rewiring** is the process of rewiring certain edges in the graph. The aim here is to enhance the graph in some form to help improve the performance of the Graph Neural Network. Graph rewiring is used to counter issues like homophily and overquashing in GNNs. Specifically, Deep Heterophily Graph Rewiring(DHGR) [39] focuses on rewiring the graph using label distribution. Heterogeneous graph rewiring approach(HDHGR) [18] uses a meta-path induced method to determine the similarity in heterogeneous graphs to boost the performance of HGNNs and, Stochastic Discrete Ricci Flow [19] uses a curvature-based method to rewire the graph and avoid over squashing.

3. **Graph Sampling** is another strategy to help enhance graph performance through augmentation. Here a given graph G is passed through a sampler to get a smaller subgraph. Often samplers can be of various types, such as vertex-based or edge-based samplers. The SUBG-CON [20] attempts to sample graphs using the importance scores of neighbors and samples some nodes to preserve context. This is very similar to a vertex-based sampler. Similarly, methods like NeuralSparse [40], use a

parameterized method to study structural and non-structural information in a graph and remove outside edges based on this information. There are also more advanced methods like Metropolis-Hastings Algorithm [21] to sample from a graph using the Markov Chain Distribution. Lottery Ticket Hypothesis [41] states that sub-networks have been pruned accurately and can be trained again to attain similar performance as the original deep neural networks are large in size.

4. **Node Dropping** focuses on dropping nodes using a node mask. As a part of this process, nodes and edges associated with that node are dropped from the given graph.

5. **Node Insertion** [22] adds new nodes and edges to the given graph. These are also known as virtual nodes.

Graph augmentation has a lot of applications and breakthroughs in two major areas, namely Graph Contrastive Learning(GCL) and adversarial defense. Below, we give some details on how graph augmentation is used in both of these areas.

1. Graph Contrastive Learning(GCL) focuses on augmenting a given graph and viewing the augmented pair as positive pairs and the rest as negative pairs. Multiple methods in this area differ in their augmentation strategy. For example, Graph Contrastive Coding [42] uses graph sampling as an augmentation strategy. The methodology here goes node-by-node and samples 2-hop subgraphs for each node, treated as the *central node*. Each subgraph is used to generate an embedding for the central node. Finally, the contrastive loss helps ensure that embeddings generated for the same central node are similar across all subgraphs. Therefore, GCC helps pre-train graphs with *subgraph instance discrimination* and uses *graph sampling* as a data augmentation method. Some GCL methods use multiple structural augmentations and try to preserve similarities in all the augmentation strategies. For example, [14] uses multiple structural augmentation methods like node dropping, edge perturbation, attribute masking, and subgraph sampling. The contrastive loss function tries to maximize the node embedding agreement between all four augmentation strategies for a given graph.

2. Adversarial attacks are usually conducted by augmenting the graph in an unnoticeable way such that the performance of the model drops. Usually, the attacker aims to satisfy a constraint to make the attack unnoticeable so that the performance of the model drops. The Projected Gradient Descent(PGD) Topology Attack [43] is an adversarial method that uses graph augmentation. This method aims to find a symmetric matrix S where $\mathbf{S} \in \{0, 1\}^{N \times N}$ that can be used to attack the given graph. The edges indicated with one in S are dropped, and the others are retained. To do this, we find $S \in [0, 1]$, where S would consist of continuous values indicating

the probability of dropping a given edge. This helps transform a discrete optimization problem into a continuous optimization problem that can be solved using the projected gradient descent(PGD) method. Additionally, augmentation is also used to mitigate adversarial impacts in graphs. This is predominantly done using graph rewiring. For example, methods like GNNGuard [44] and G-Jaccard [45] help prune edges between nodes that have feature dissimilarities beyond a threshold.

Chapter 3

Methodology

Our approach attempts to identify the spectral point of a subgraph with high Micro F1 and Macro F1 scores and augment our larger graph towards the same. Towards this front, we have developed algorithms that help augment the graph to the desired spectral point by altering some geometrical properties. Furthermore, we apply GraphSAGE to produce embeddings on the original graph and perform node classification on both these graphs. The following sections give a brief background on GraphSAGE and Spectral Moments, as shown in Section 3.1, to help explain these concepts in detail as they are central to our work. We then emphasize the methodology used to interpolate surface plots from a sampled set of spectral moments and their accuracies and visualize these plots as shown in3.2. Finally, we propose our method to augment the current spectral moment($[m_2, m_3, m_4]$) to a desired spectral moment($[m'_2, m'_3, m'_4]$) as shown in 1, by providing simple and efficient methods to efficient to increase and decrease the second, third and fourth moment.

3.1 Background

3.1.1 GraphSAGE

GraphSAGE [10] aims to generate node embeddings in large-scale graphs. By using the AGGREGATE and UPDATE functions, GraphSAGE can effectively capture the local structure of a node's neighborhood and generate meaningful embeddings that capture the node's position in the graph.

It uses two operations for generating node embeddings: AGGREGATE and UPDATE. The AGGREGATE function is responsible for collecting messages from the neighbors. The UPDATE function concatenates this with the node's embedding to produce a final embedding for the node. GraphSAGE also uses a graph sampling method to sample a set of neighbors for a given node instead of using all its neighbors. The following Equation 3.1 represents the AGGREGATE and UPDATE steps, respectively. Here v represents a node, $N(v)$ represents all of v's neighbors, h_k^i represents node embeddings of any node/nodes i, W^k represents a weight matrix, CONCAT refers to the concatenation operation in vectors.

$$\begin{aligned} h_{N(v)}^k &\leftarrow \text{AGGREGATE}\left(\{h_u^k, \forall u \in N(v)\}\right) \\ h_v^k &\leftarrow \sigma\left(W^k \cdot \text{CONCAT}(h_v^{k-1}, h_{N(v)}^k)\right) \end{aligned} \tag{3.1}$$

3.1.2 Spectral Moments

Let $G = (V, E)$ be a graph, where V is the set of vertices $V = v_1, v_2, ...v_n$ and E $\subseteq V \times V$ is the set of edges. Let $A \in R^{n \times n}$ denote the adjacency matrix of G, defined as $A_{(i,j)}$ = 1 if there exists an edge between node i and j, and 0 otherwise. Let D be the diagonal degree matrix of the given graph, where $D_{i,i}$ = $\sum_{(i,j)\in \text{E}} A_{i,j}$. The *Normalized Laplacian* of the graph is defined as shown in Equation 3.2:

$$L = I - D^{-1/2} A D^{-1/2} \tag{3.2}$$

The spectrum of a graph has connections to its properties.

The eigenvalues of the *Normalized Laplacian* is its spectrum as with 0 = $\mu_1 \leq \mu_2 ... \mu_{n-1} \leq \mu_n$ = 2, where μ_i represents the i^{th} eigenvalue. We use P to find the spectrum of graph G as shown in [1].

This is defined as the expectation on the l^{th} power of the random walk transition matrix's eigenvalues as shown in Equation 3.3, where m_l refers to the lth spectral moment. Spectral moments can also capture the shape of the distribution and is closely related to spectral density.

$$m_l = 1/n \sum_{i=1}^{n} \lambda_i^l \tag{3.3}$$

Here the random walk transition matrix is defined as shown in Equation 3.4

$$P = D^{-1}A \tag{3.4}$$

Our work focuses on augmenting graphs using their spectral moment. For this, we take the spectral moment of the current graph to the spectral moment of a graph with high node classification accuracy. For this, we refer to the results of Jin et al. 3.3 to understand how to modify the spectral moments of an input graph. As mentioned in Jin et al., we use the spectral moments of the random walk transition matrix over other methods that use the combinatorial Laplacian matrix because we want compact embeddings that are not bound by the graph size.

The following sections elucidate further on the *second*, *third*, and *fourth* spectral moment and their relations to geometrical properties in a given graph and are taken from Jin et al.

Second Spectral Moment(m$_2$)

The *second* spectral moment is directly related to the average return probability of a closed 2-walk and the average degree of a node. Therefore, one way to increase the *second* spectral moment is to add more edges to the network. The following example elucidates better.

$$m_2 = \mathbb{E}(\lambda^2) = \mathbb{E}(d_i)\mathbb{E}(\frac{1}{d_i d_j}) \tag{3.5}$$

According to the above equation, edge addition will increase the average degree of the nodes, which m_2 is directly proportional to m_2. Furthermore, decreasing the degree of nodes in the closed 2-walks can also help increase the value of m_2. Conversely, if we want to decrease the *second* spectral moment, we can reduce the average degree of nodes or increase the degree of nodes in a closed 2-walk in the graph.

Third Spectral Moment(m_3)

The third spectral moment directly relates to the average return probability of a closed 3-walk and the average number of 3-walks a node belongs to. As shown in Jin et.al.3.3, m_3 can be defined as follows. Here, $\mathbb{E}(\triangle_i)$ indicates the average number of triads/3-paths a node belongs to, and d_i refers to the degree of node i.

$$m_3 = \mathbb{E}(\lambda^3) = \mathbb{E}(\triangle_i)\mathbb{E}(\frac{1}{d_h d_i d_j}) \tag{3.6}$$

From equation 3.6, we infer that adding triads with a low sum of the degree of nodes into the graph can help increase the average return probability of a closed 3-walk which would boost the *third* spectral moment. Adding triads with a lower sum of degrees would reduce the values of d_i, d_j, and d_k, which is inversely proportional to the return probability of closed 3-walks. Furthermore, it can also increase the average number of triads a node is involved in, which according to equation 3.6 would increase the value of m_3. In other words, the first method to improve the m_3 is to add more triads as this will increase the number of triads a node belongs to goes up. The second method would be to increase the joint probability distribution of triads by decreasing the average degree sum of a triad in the graph, increasing the probability of a 3-walk. On the other hand, if we want to decrease the value of m_3, we can decrease the number of triangles a node belongs to or reduce the joint probability of a triad. The formula below helps describe this further.

Fourth Spectral Moment(m_4)

The *fourth* spectral moment is directly related to the average joint probability distribution of a 4-walk and the average number of edges, wedges, and squares a node belongs to the same as mentioned in Jin et.al3.3. Formally this can be defined as:

$$m_4 = \mathbb{E}(\lambda^4) = (\mathbb{E}(d_i) + 4\mathbb{E}\binom{d_i}{2} + \mathbb{E}(\square_i))\mathbb{E}(\frac{1}{d_i d_j d_k d_l}) \tag{3.7}$$

Here in Equation 3.7, $\mathbb{E}\binom{d_i}{2}$ refers to the average number of wedges a node belongs to, and $\mathbb{E}(\square_i))$ represents the average number of quadrilaterals/4-paths a node belongs to in the given graph. Edges have two 4-walks, wedges have four 4-walks, and quadrilaterals have eight 4-walks, contributing the most to the *fourth* spectral moment. Therefore increasing the average number of 4-cycles a node shares can help improve the value of m_4. We can further improve it if we reduce the degree of nodes in a closed 4-walk, increasing the return probability of the 4-walk. Conversely, we can reduce the value of m_4 by decreasing the average number of closed 4-walks per node or by having many high-degree closed 4-walks, which, as mentioned previously, reduces the return probability.

3.2 Plotting Spectral Moments

The aim is to identify the spectral moment that maximizes the Micro-F1 and Macro-F1 scores for node classification. As it is challenging to sample every subgraph in the space, one can easily miss out on the subgraph that maximizes the accuracy. Therefore, we need to interpolate two things after sampling subgraphs from the given graph, namely:

- A surface plot of m_2, m_3, and m_4 that is continuous, representing a majority of subgraphs of a specified sampling percentage.

- The node classification Micro-F1 and Macro-F1 scores of the spectral moments on the surface plot using the same sampled set of subgraphs as above.

These F1 scores then colorize the surface plot to produce a smooth surface plot that is easy to visualize.

The first step would be to sample subgraphs from a given graph. We use the random node sampler and vary the sampling ratio between 10, 50, and 90 percent. We also use a 10 percent sample on the entire graph that acts as the test set, with no overlap with the train subgraphs sampled. The random node sampler generates subgraphs for each of the proportions mentioned. We pass the subgraphs through a 2-layer GraphSAGE model. We obtain the Micro-F1 and Macro-F1 scores for node classification across the sampled subgraph for each graph.

The next step is to fit a surface for m_2, m_3, and m_4 for each subgraph. To do this, we use a *linear interpolator*. *Interpolation* is estimating unknown data points between known points. Most interpolation methods learn a function $y_i = f(X_i)$. We use the *linear interpolator* that fits a linear polynomial between every pair of points. This way, by constructing a grid for m_2 and m_3, we can interpolate the values of m_4 and produce a surface plot.

The final step is to interpolate the Micro-F1 score and Macro-F1 score of node classification of the spectral moments in the surface plot and colorize the surface with this. We use scattered interpolant in MATLAB, which uses the Delaunay Triangulation method [46]. The Delaunay triangulation creates a set of non-overlapping triangles that cover the convex hull of the scattered data points. The cubic interpolant is computed separately within each triangle. The boundary conditions are set such that the interpolant is smooth and has zero-second derivatives at the boundary of the convex hull. The Delaunay triangulation ensures that the cubic interpolant is continuous across the edges of adjacent triangles, resulting in a smooth and well-behaved interpolant for the scattered data.

3.3 Modify Spectral Moments

The following sections detail how we modify the spectral moments of a given graph. We first summarize the algorithm3.3.1 that helps augment a given graph to a desired spectral moment. This process usually involves changing the graph structure to help increase/decrease the values m_2, m_3, and m_4. The following sections, namely Section 3.3.2 and Section 3.3.3 talk about our approach to increase and decrease these moments by augmenting the graph structure. Finally, to conclude, this section gives an overall summary of our approach.

3.3.1 Algorithm For Modifying Spectral Moments

The overall algorithm for augmenting the graph using its spectral moments is provided in Algorithm 1. The algorithm takes a graph(G) and the desired spectral moment, where the m_2, m_3, and m_4 are at indices at zero, one, and two, respectively. The methodology can briefly be described as follows: First, we find the spectral moment of the graph. These moments are stored in $desiredMoment$, where the m_2, m_3, and m_4 are indexed at zero, one, and two, respectively. Second, index-wise, we find the difference between the current and desired spectral moments to obtain the difference in m_2, m_3, and m_4. If we obtain a difference $>$0 for any of these values, we must increase the corresponding moment(s). Similarly, if the difference $<$0, we must reduce the corresponding moment(s). To ensure that the spectral moments of the current graph do not get further from the desired moments, we have a variable that stores the best graph with the closest spectral moment to the desired up until now and an early stopping condition that counts every time the moments get worse. Our methodology converges the graph's spectral moment to come within δ of the desired spectral moment or when we reach the early stopping condition(θ). The algorithms to increase and decrease spectral moments that are used in 1 are given in Section 3.3.2 and Section 3.3.3.

Algorithm 1 ModifyGraph

```
Input  : $G$, $desired\_moment$,$\delta$,$\theta$
Output: $bestGraph$
$stopping \leftarrow 0$
$current\_moments \leftarrow$ getSpectralMoments($G$)
$bestDistance \leftarrow \infty$
$prevDistance \leftarrow \infty$
$bestSpectral \leftarrow$ None
$bestGraph \leftarrow$ G
while $\|desired\_moment - current\_moment\|_2 > \delta$ and $stopping < \theta$ do
    if $bestDistance > \|desired\_moment - current\_moment\|_2$ then
        $bestDistance = \|desired\_moment - current\_moment\|_2$
        $bestSpectral = current\_moment$
        $bestGraph = G$
    if $prevDistance < \|desired\_moment - current\_moment\|_2$ then
        $stopping = stopping + 1$
    if $desired\_moment[2] > current\_moment[2]$ then
        $G =$ increaseMoments($G, m_2$)
    else if $desired\_moment[2] < current\_moment[2]$ then
        $G =$ decreaseMoments($G, m_2$)
    if $desired\_moment[3] > current\_moment[3]$ then
        $G =$ increaseMoments($G, m_3$)
    else if $desired\_moment[3] < current\_moment[3]$ then
        $G =$ decreaseMoments($G, m_3$)
    if $desired\_moment[4] > current\_moment[4]$ then
        $G =$ increaseMoments($G, m_4$)
    else if $desired\_moment[4] < current\_moment[4]$ then
        $G =$ decreaseMoments($G, m_4$)
    $current\_moments \leftarrow$ getSpectralMoments($G$)
return $bestGraph$
```

3.3.2 Algorithms to increase spectral moments

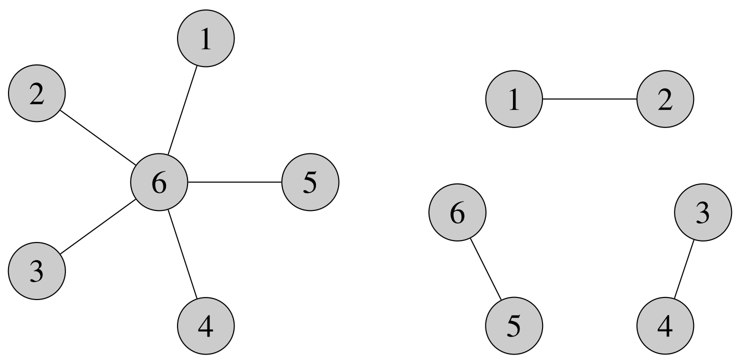
As discussed in 3.1.2, we typically have two ways of boosting the l^{th} spectral moment of a given graph, which is to either increase the average number of closed walks of length l a node is in or increase the expected probability of a closed random walk of length l.

Algorithm 2 augments the graph by constructing more closed walks of length l. This is achieved by removing the node with the highest degree along with its edges and neighbors from the input graph G and randomly creating walks of length l.

Algorithm 2 INCREASEMOMENTS

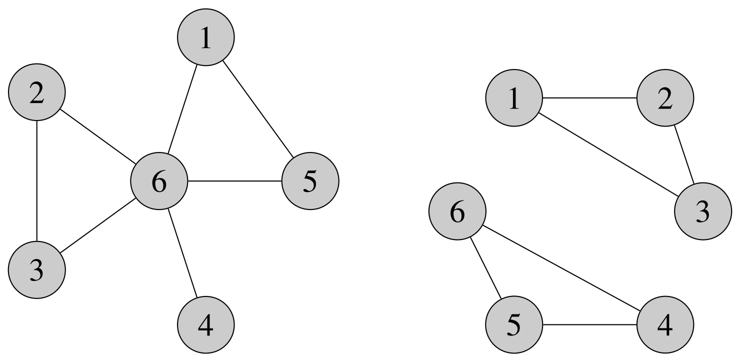
```
Input  : $G$, mode
Output: $G$
$node \leftarrow$ HighestDegreeNode($G$)
$reconstructionNodes \leftarrow$ node $\cup$ Neighbors($node$)
$G \leftarrow G - reconstructionNodes$
if mode == '$m_2$' then
    for $node_1, node_2 \leftarrow reconstructionNodes$ do
        if G.degree($node_1$) $<$ 1 and G.degree($node_2$) $<$ 1 then
            G.addEdge($node_1$,$node_2$)
if mode == '$m_3$' then
    for $node_1, node_2, node_3 \leftarrow reconstructionNodes$ do
        if G.degree($node_1$) $<$ 2 and G.degree($node_2$) $<$ 2 and G.degree($node_3$) $<$ 2 then
            G.addTriad($node_1$,$node_2$,$node_3$)
if mode == '$m_4$' then
    for $node_1, node_2, node_3, node_4 \leftarrow reconstructionNodes$ do
        if G.degree($node_1$) $<$ 2 and G.degree($node_2$) $<$ 2 and G.degree($node_3$) $<$ 2 and
          G.degree($node_4$) $<$ 2 then
            G.addSquare($node_1$,$node_2$,$node_3$,$node_4$)
return G
```

We know that m_2 is directly proportionate to $\mathbb{E}(\frac{1}{d_i d_j})$. Therefore in the case of m_2, we just add more independent edges, which would increase the value of $\mathbb{E}(\frac{1}{d_i d_j})$. In Algorithm 2, this is done by finding the node with the highest degree, removing it along the neighboring nodes from the graph, and adding independent edges to the graph. This can be understood by referring to Figure 3.1. Here, in (a), when we calculate $\mathbb{E}(\frac{1}{d_i d_j})$, we would go through each two-walk and find the average $\frac{1}{d_i d_j}$. Since we have five edges associated with node six(d_6 = 5), we would be dividing by five multiple times as we average $\frac{1}{d_i d_j}$ across each two-walk associated with node 6. This will lower the $m2$. To increase this, we can evenly distribute the edges associated with node six across all the other nodes as shown in (b), thereby increasing the value of $\mathbb{E}(\frac{1}{d_i d_j})$.



(a) Before Augmentation (b) After Augmentation

Figure 3.1: Example of increasing the second spectral moment.



(a) Before Augmentation (b) After Augmentation

Figure 3.2: Example of increasing the third spectral moment.

In the case of m_3, we add closed 3-cycles or triangles to the given graph using the same approach as m_2. Here, we find the node with the highest degree, remove it along with its neighboring nodes, and construct triangles using these removed nodes such that the triangles are independent. Refer to Figure 3.2 to help understand this further. To find $\mathbb{E}(\frac{1}{d_i d_j d_k})$ we iterate through every closed 3-path in the graph, find the value of $\frac{1}{d_i d_j d_k}$ for each of them and average the values. As shown in Figure 3.2(a), the degree of node 6 is five($d_6 = 5$), and there are two triads associated with it as well. Therefore, we would divide by five multiple times as we average $\frac{1}{d_i d_j d_k}$ across every 3-walk associated with node 6. As a result, the value of m_3 would be low. To *increase* the value of m_3, we can evenly distribute the degree of node six and increase the number of triads across other nodes as shown in Figure 3.2(b). This helps increase the value of $\frac{1}{d_i d_j d_k}$, because we would

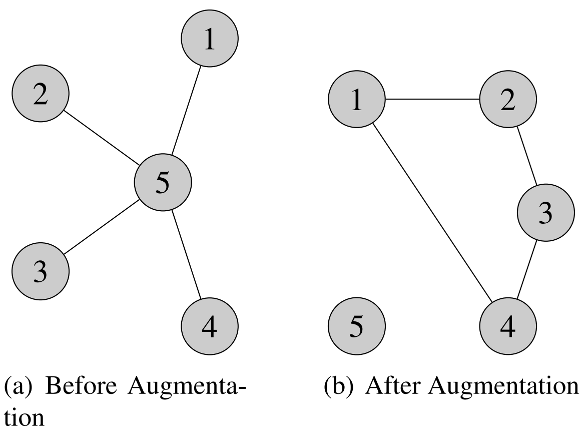
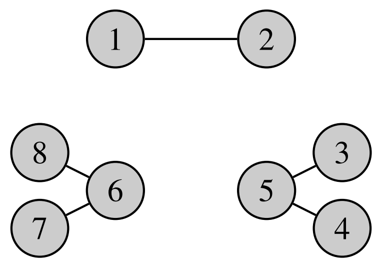


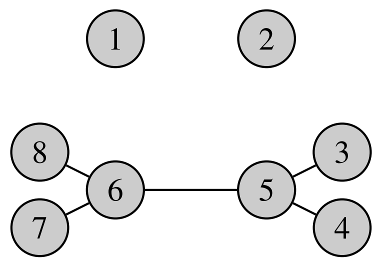
Figure 3.3: Example of increasing the fourth spectral moment.

not be dividing by a high degree multiple times, thereby increasing the value of m_3, while improving the value of $\mathbb{E}(\triangle_i)$ as well.

Finally, in the case of m_4, we add closed four cycles or quadrilaterals to the given graph using the same approach as in m_2 and m_3. Here, we find the node with the highest degree, remove it along with its neighboring nodes, and construct quadrilaterals using these removed nodes such that they are independent. We know that m_4 is directly proportional to the value of $\frac{1}{d_i d_j d_k d_l}$ as shown in Section 3.1.2. Refer to Figure 3.3 to understand this better. To find $\mathbb{E}(\frac{1}{d_i d_j d_k d_l})$ we iterate through every closed 4-path in the graph, for which we find the value of $\frac{1}{d_i d_j d_k d_l}$ and average these values. As shown in Figure 3.3(a), the degree of node 5 is four($d_5 = 4$), and there are many edges and wedges associated with it as well. From 3.1.2, we know that m_4 is also directly proportional to the number of edges and wedges in the graph. Therefore, we would divide by four multiple times as we average $\frac{1}{d_i d_j d_k d_l}$ across every 4-walk associated with node 5. causing the value of m_4 to be low. To *increase* the value of m_4, we can evenly distribute the degree of node five and increase the number of quadrilaterals across other nodes as shown in Figure 3.3(b). This helps increase the value of $\frac{1}{d_i d_j d_k d_l}$, because we would not be dividing by a high degree multiple times, thereby increasing the value of m_3, while improving the value of $\mathbb{E}(\square_i)$ as well.



(a) Before Augmentation



(b) After Augmentation

Figure 3.4: Example of decreasing the second spectral moment.

3.3.3 Algorithms to decrease spectral moments

As discussed in 3.1.2, we typically have two ways of decreasing the l_{th} spectral moment of a given graph, which is to either reduce the average number of closed walks of length l a node is in or the average return probability of a walk of length l in the graph. Algorithm 3 augments the graph by further increasing the degree of each node an already high degree sum closed walk of length l. In the case of m_2, this is done by connecting 2 high-degree unconnected nodes to create closed 2-walk, and removing a low degree 2-walks, in every iteration. In the case of m_l for $l > 2$, the methodology can be generalized as finding the closed l-walk with the lowest degree sum($n_1 \rightarrow n_2 \rightarrow \ldots n_l \rightarrow n_1$), removing the edges associated to *low degree sum walk*, finding the closed l-walk with highest degree sum($n'_1 \rightarrow n'_2 \rightarrow \ldots n'_l \rightarrow n'_1$), and finally, connecting the nodes in the *low degree sum walk* to the nodes in the *high degree sum walk*s($n_1 \rightarrow n'_1, n_2 \rightarrow n'_2 \ldots n_l \rightarrow n'_l$).

These methods are further elucidated with examples for m_2,m_3, and m_4. Figure 3.4 is an example that helps explain how to reduce m_2. Here, the algorithm first finds the edge with the *lowest sum of the degree of nodes* as shown in 3.4(a), which is node **1** and node **2**. This

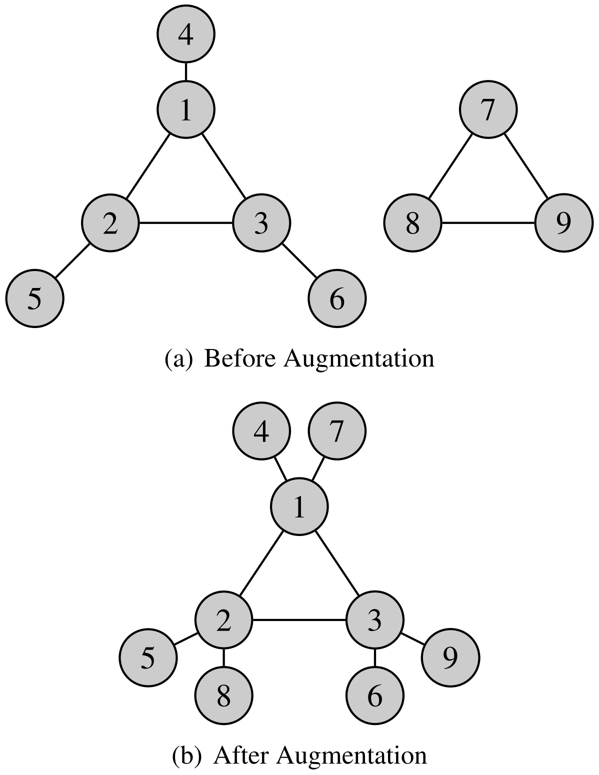


Figure 3.5: Example of decreasing the third spectral moment.

edge is removed from the graph and is used to connect the high-degree nodes, which n this case is node **5** and node **6** as shown in 3.4(b).

The complexity of this operation is $O(N^2)$. We must parse the graph twice for each node to find the two unconnected nodes. Regarding m_3 and m_4, this approach's complexity will increase to $O(N^3)$ and $O(N^4)$ due to the complexity of finding 3 and 4 unconnected components, respectively.

We solve this by finding the fundamental set of cycles [47] and filtering to find the same to find the *lowest and highest degree sum* closed walk of size l. The *lowest degree sum cycle* is removed, and those edges are added to the *highest degree sum walk*.

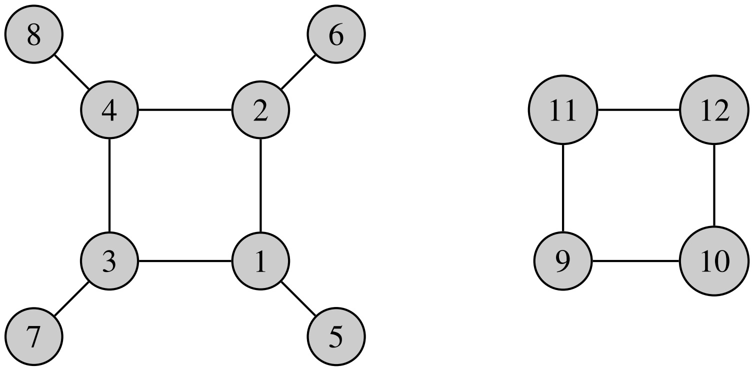
For m_3, Algorithm 3 helps create *high degree sum* triads and get rid off low degree sum triads, which reduces m_3. For example, refer to Figure 3.5. Here, the low degree sum triad is **7** $\rightarrow$ **8** $\rightarrow$ **9**. The high degree triad is indicated as **7** $\rightarrow$ **8** $\rightarrow$ **9** as shown in 3.5(a). After

Algorithm 3 DECREASEMOMENTS

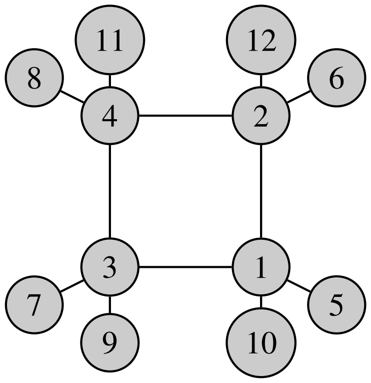
```
Input  : $G$, mode
Output: $G$
if mode == '$m_2$' then
    $minEdge \leftarrow getEdge(G)$
    $maxNode_1, maxNode_2 \leftarrow getUnconnectedNodes(G, 2)$
    G.removeEdge($minEdge$)
    G.addEdge($maxNode_1, maxNode_2$)
if mode == '$m_3$' then
    $minNode_1, minNode_2, minNode_3 \leftarrow$ min($G$.getCycle($3$))
    $maxNode_1, maxNode_2, maxNode_3 \leftarrow$ max($G$.getCycle($3$))
    G.removeTriad($minNode_1, minNode_2, minNode_3$)
    for i in range(1,4) do
        G.addEdge($minNode_i, maxNode_i$)
if mode == '$m_4$' then
    $minNode_1, minNode_2, minNode_3, minNode_4 \leftarrow$ min($G$.getCycle($4$))
    $maxNode_1, maxNode_2, maxNode_3, maxNode_4 \leftarrow$ max($G$.getCycle($4$))
    G.removeTriad($minNode_1, minNode_2, minNode_3, minNode_4$)
    for i in range(1,5) do
        G.addEdge($minNode_i, maxNode_i$)
return $G$
```

applying m_3 augmentation, we would end up removing the edges between nodes **7**, **8** and **9** and connecting nodes **7** $\rightarrow$ **1**, **8** $\rightarrow$ **2** and **9** $\rightarrow$ **3** 3.5(b). This increases the degree of each node **1**, **2** and **3** and further reduces the number of triads in the given graph, leading to a decrease in m_3.

Similarly, Algorithm 3 helps create *high degree sum* quadrilaterals and eliminate low degree sum quadrilaterals, which reduces the value of m_4. For example, refer to Figure 3.6. Here the low degree quadrilateral is **9** $\rightarrow$ **10** $\rightarrow$ **12** $\rightarrow$ **11** as shown in Figure 3.6(a). The high degree quadrilateral is **1** $\rightarrow$ **2** $\rightarrow$ **4** $\rightarrow$ **3**. After applying m_4 augmentation, we would end up removing the edges between nodes **9**, **10**, **12** and **11** and connecting nodes **10** $\rightarrow$ **1**, **12** $\rightarrow$ **2**, **11** $\rightarrow$ **4** and **9** $\rightarrow$ **3** as shown in 3.6(b). This increases the degree of each node **1**, **2**, **3** and **4** and further reduces the number of quadrilaterals in the given graph, leading to a decrease in m_4.



(a) Before Augmentation



(b) After augmentation

Figure 3.6: Example of decreasing the fourth spectral moment.

3.4 Summary

Our approach aims to augment the graph using a desired spectral moment to maximize the Micro-F1 score and Macro-F1 score for node classification tasks on the graph. To do this, we first have to identify a desired spectral moment. This can be done by interpolating a surface of m_2, m_3, and m_4 and colorizing them with micro-F1 and macro-F1 scores from a sample of subgraphs as mentioned in Section 3.2. From these plots, we can identify the desired spectral moment that maximizes both micro-F1 and macro-F1 scores and is used as the desired spectral moments. Finally, we augment the graph towards the desired spectral moment using Algorithm 1 2 and 3. We employ the GraphSAGE model to evaluate the augmentation method's performance. Here we pass the original graph and the augmented graph through GraphSAGE and obtain the Micro-F1 score and Macro-F1 score for both of them. We use no edge sampling to avoid any further perturbations to the spectral moments and use the mean aggregator as this closely resembles other models like GNNs and GCNs. The following chapter highlights the experiments conducted and their results in further detail.

Chapter 4

Experiments

In this chapter, we broadly talk about the datasets used in our experiments, our experimental setup, and the results obtained from the experiments. We use the following datasets for our evaluation Cora [48], Citeseer [49], Pubmed [50], Protein-Protein interaction [10], and the Facebook dataset [51] which have found a lot of usage in the area of graph representational learning. We can find details about these datasets in Section 4.1. In Section 4.2, we talk about GraphSAGE, the data sampling strategy used for generating spectral moments and training our model, and the parameter configuration used. Finally, Section 4.4 details the results obtained as a part of our experiments, which include the surface plots of spectral moments that extrapolate continuous representations of the Micro-F1 and Macro-F1 scores of node classification for different values of m_2, m_3 and m_4, the change in Micro-F1 and Macro-F1 scores before and after augmenting the graph using it's spectral moments and the graph sparsification that occurs due to the augmentation.

4.1 Datasets

This section briefly describes the datasets we use for our experiments. The datasets we use fall into the categories of Citation Networks(CORA, PubMed, and Citeseer), Protein-protein interaction Networks(PPI), and Social Networks(Facebook). Some details about these datasets are listed below:

1. **Cora**: The Cora dataset [48] consists of Machine Learning papers with directed links representing citations between various papers. These papers are classified into

one of the following seven classes: Case-Based, Genetic Algorithms, Neural Networks, Probabilistic Methods, Reinforcement Learning, Rule Learning, and Theory. There are 2,708 scientific publications(nodes) with 5,429 links between the papers. Each publication in the dataset is described by a 0/1-valued word vector of size 1,433, indicating the absence/presence of the corresponding word from the dictionary, which acts as the feature vector for each node in the graph.

2. **Citeseer**: The Citeseer dataset [49] consists of a collection of publications in various domains in Computer Science. These papers are classified into one of the six classes, which include Agents, Artificial Intelligence(AI), Databases(DB), Information Retrieval (IR), Machine Learning(ML), and Human-Computer Interaction(HCI). The value indicated in the brackets are the class labels given in the dataset. Citeseer consists of 3,312 scientific publications in one of the above six labels. The citation network consists of 4,732 links. Again this dataset consists of a 0/1-valued word vector of size 3,703 indicating the absence/presence of the corresponding word from the dictionary, which acts as the feature vector for each node in the graph.

3. **Pubmed**: The Pubmed dataset [50] consists of a collection of scientific publications about Diabetes. These papers are classified into one of three categories which include Diabetes Mellitus, Experimental(0), Diabetes Mellitus Type 1(1), and Diabetes Mellitus Type 2(2). The value indicated in the brackets are the class labels given in the dataset. Pubmed consists of 19,717 nodes put into one of the above three labels. The network consists of 44,338 links, where each link indicates a citation. The dataset consists of TF/IDF vectors representing every publication in the given dataset, built using a dictionary of 500 unique words.

4. **Protein Protein Interaction(PPI)**: The protein-protein interaction dataset [10] consists of proteins as nodes, and the edges represent the interaction between these proteins. The dataset has multiple labels where each node can belong to more than one

class. Here, the labels are constructed using properties such as positional gene sets, motif gene sets, and immunological signatures as features and gene ontology sets as labels for each node. Finally, the PPI dataset consists of many subgraphs that are not connected. There are 20 graphs, with an average of 2,373 nodes and an average degree of 28.8 per node in each graph. We treat this as one single graph. Finally, the PPI dataset is also very sparse in features, where about 42 percent of the nodes have no features. Therefore, leveraging neighborhood structure more usefully for better node classification becomes essential.

5. **Facebook**: The Facebook dataset [51] is a page-page graph consisting of verified facebook pages. Each page represents a node, and the edge between each page is mutual. The graph consists of 22,470 nodes and 171,002 edges connecting nodes. The dataset belongs to the category of multi-class classification. The labels used here include politicians, governmental organizations, television shows, and companies, where each node belongs to one of these classes. Lastly, each web page has a feature vector obtained by parsing the respective web page.

4.2 Experimental Setup

The following section details the experimental setup used to run the model. The following sections elucidate the parameters used to estimate the spectral moments of a graph, the steps used to interpolate the surface plots, the parameters of the GraphSAGE architecture used, and the parameters used in the augmentation algorithms.

Spectral moments

In our work, we use the APPROXSPETRALMOMENT Algorithm [52] to approximate the spectral moments of the graph. The algorithm takes many random walks and finds the probability of a random walk of length l. This approximated value becomes the l^{th} spectral moment. We set the number of random walks, s, as 10,000 and l as two, three, and four. We use the code provided by Jin et al. [1].

Surface plot interpolation

As mentioned in Section 3.2, we need to sample subgraphs to extrapolate the surface plot, where we construct a surface using m_2, m_3 and m_4 and colorized using the Micro-F1 score and Macro-F1 score. Therefore, two plots are generated for the set of moments. The following steps are taken to extrapolate the surface plot. We sample subgraphs of size 10 percent, 50 percent, and 90 percent using the random node sampling method, which takes a random collection of nodes and draws edges, and takes the induced subgraph based on these nodes. We sample a uniform 10 percent test set exclusive to the training subgraphs sampled on which we find the F1 scores. The model used here is GraphSAGE, as explained in the following section. To plot the surface, we use linear interpolation. We use Delauney Triangulation to interpolate the F1 scores. The surface plots are constructed using MATLAB, allowing our plots to be very interpretable.

GraphSAGE Architecture

For the GraphSAGE convolutional operator, we use the mean aggregator which is explained in Section 3.1.1. We do not sample neighbors while performing aggregation and use the entire graph to ensure that the spectral moment of the graph we train on does not alter. Our model consists of two GraphSAGE convolutional layers with a hidden layer of dimension 64×64 for Cora, Citeseer, PubMed, and Facebook. For PPI, we use two GraphSAGE convolutional layers with a hidden layer of dimension 512×512. We use the ADAM optimizer and train for 100 epochs. We use a uniform 10 percent test set to obtain Micro-F1 and Macro-F1 scores.

Parameters for Augmentation

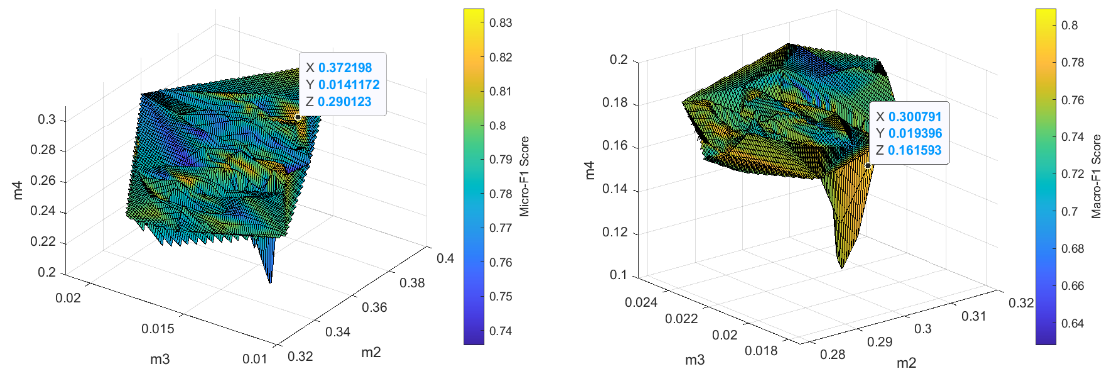
Algorithm 1 has two parameters δ and θ, where the former can be applied to measure L2-NORM, and we use the latter as the stopping criterion, which is used to perform early stopping. We set the values of δ and β to 0.01 and 10, respectively.

4.3 Surface plots

The following are the surface plots of the subgraphs. For each dataset, we plot a surface for each graph sampling ratio, namely 10, 50, and 90 percent of nodes. Furthermore, each surface is colored by Micro-F1 and Macro-F1 scores of node classification on each dataset. We identify in each plot the spectral moment with the highest Micro-F1 and Macro-F1 scores. As a part of this section, we give an example plot per dataset for the Micro F1 score and Macro F1 score. We also show the highest F1 score points on each plot. Note that while they need not be the same in this section, we select points with high F1 scores on both plots for our augmentation process.

In Figure 4.1, we have the surface plot for the Cora dataset. In plot 4.1(a), the sample size taken for the interpolating Micro-F1 score is 50 percent, and in plot 4.1(b), it is 90 percent. As shown in the diagram, the spectral moment with the highest Micro F1 score is $m_2 =$ 0.372, $m_3 =$ 0.0141, and $m_4 =$ 0.290, and that with the highest Macro F1 score is $m_2 =$ 0.30, $m_3 =$ 0.0193 and $m_4 =$ 0.161.

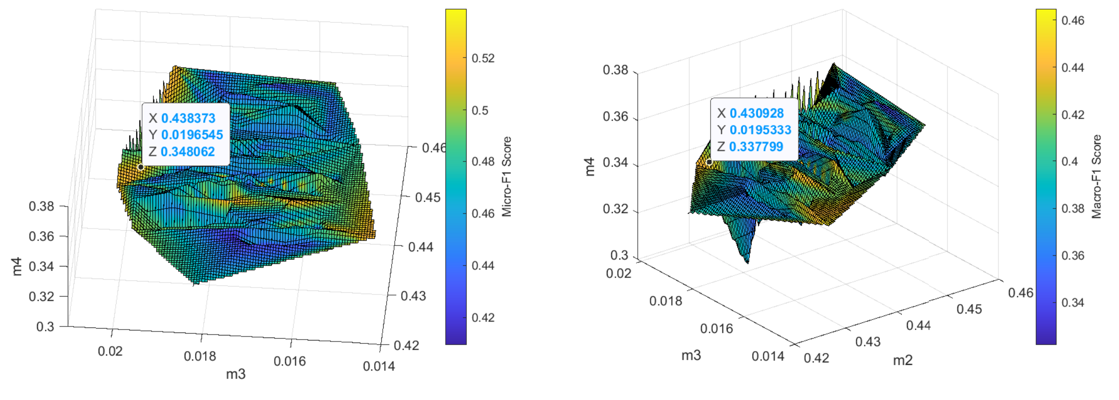
Figure 4.2 shows the surface plot for the Citeseer dataset. In both plots 4.2(a) and 4.2(b), the sample size taken for the interpolating Micro-F1 score is 90 percent. As shown in the diagram, the spectral moment that gives the highest Micro-F1 score here is $m_2 =$ 0.438, $m_3 =$ 0.019, and $m_4 =$ 0.348 and that which gives the highest Macro-F1 score is $m_2 =$ 0.430, $m_3 =$ 0.019, and $m_4 =$ 0.337. The points are very close because the sample size used was the same.



(a) Micro F1 for 50 percent sample size

(b) Macro F1 for 90 percent sample size

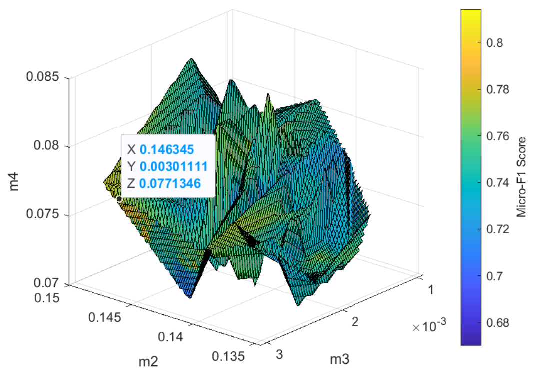
Figure 4.1: Surface plots for Micro F1 score and Macro F1 score in **Cora**, on **50**, and **90** percent subgraph samples. Better viewed in color.



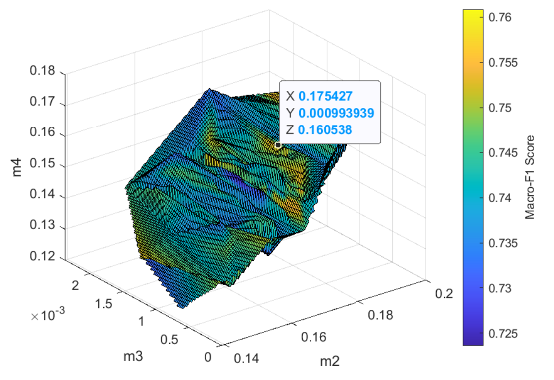
(a) Micro F1 for 90 percent sample size

(b) Macro F1 for 90 percent sample size

Figure 4.2: Surface plots for Micro F1 score and Macro F1 score in **Citeseer**, on **90** percent subgraph samples for both. Better viewed in color.



(a) Micro F1 for 90 percent sample size



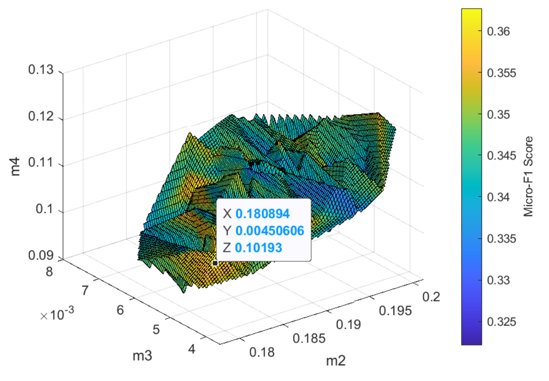
(b) Macro F1 for 10 percent sample size

Figure 4.3: Surface plots for Micro F1 score and Macro F1 score in **Pubmed**, on **90** and **10** percent subgraph samples respectively. Better viewed in color.

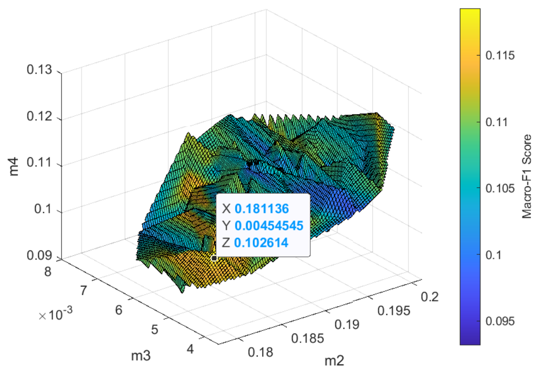
In Figure 4.3, we have the surface plot for the Pubmed dataset. In plot 4.3(a), the sample size taken for the interpolating Micro-F1 score is 90 percent, and in plot 4.3(b), we take a 10 percent sample. As shown in the diagram, the spectral moment here that maximizes the Micro-F1 score is m_2 = 0.146, m_3 = 0.003, and m_4 = 0.077. Similarly the spectral moments that maximize the Macro-F1 score is m_2 = 0.175, m_3 = 0.0009, and m_4 = 0.160.

In Figure 4.4, we have the surface plot for the PPI dataset. In plots 4.4(a) and 4.4(b), the sample size taken for the interpolating Micro-F1 score is 10 percent. As shown in the diagram, the spectral moment that maximizes the micro F1 score is m_2 = 0.18, m_3 = 0.0045, and m_4 = 0.101. Similarly, the moments that maximize the macro F1 score is m_2 = 0.181, m_3 = 0.0045, and m_4 = 0.1026. Note that the similarity in moments is likely due to the subgraph sample size being the same.

In Figure 4.5, we have the surface plot for the Facebook dataset. In plots 4.5(a) and 4.5(b), the sample size taken for the interpolating Micro-F1 score and the Macro-F1 scores are both 10 percent. As shown in the diagram, the spectral moment used to maximize the value of the Micro-F1 score is m_2 = 0.230, m_3 = 0.013, and m_4 = 0.196. Similarly, the

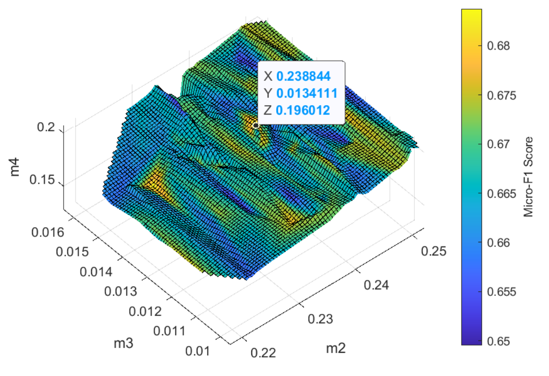


(a) Micro F1 for 10 percent sample size

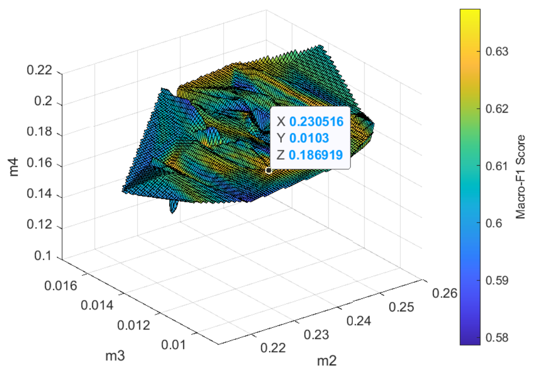


(b) Macro F1 for 10 percent sample size

Figure 4.4: Surface plots for Micro F1 score and Macro F1 score in **PPI**, on **10** percent subgraph samples for both. Better viewed in color.



(a) Micro F1 for 10 percent sample size



(b) Micro F1 for 10 percent sample size

Figure 4.5: Surface plots for Micro F1 score and Macro F1 score in **Facebook** on **10** percent subgraph samples for both. Better viewed in color.

spectral moment used to maximize the Macro-F1 is $m_2 = 0.230$, $m_3 = 0.0103$, and $m_4 = 0.180$.

4.4 Results after Spectral Augmentation

The following section details the results obtained post-augmentation, namely from the perspective of the accuracy of node classification and graph sparsification. Briefly, in some instances, the performance of node classification of the graph embedding method improves post-augmentation. Furthermore, we also see that the graph gets significantly sparsified in many cases while holding onto the same accuracy level. Section 4.4.1 highlights the changes in accuracy before and after modification across the five datasets. Section 4.4.2 talks about the changes in sparsity after augmenting a given graph.

4.4.1 Changes in F1-scores

This section highlights the changes in F1-Score in node classification for the datasets considered. Here, the graph is first classified using a layered GraphSAGE neural network. The graph is then augmented on its spectral moments and classified again.

Refer to Table 4.1 for changes in Micro-F1 and Macro-F1 scores before and after augmentation. The columns indicate the change in spectral moment. For example, 90 $\rightarrow$ 10, is augmenting a 90$\%$ subgraph to the spectral point of of the highest performing 10$\%$ subgraph. In each cell, the first value is the (Micro-F1, Macro-F1) before augmentation. The second value is the (Micro-F1, Macro-F1) after augmentation. Finally, the third value is the percentage increase in the F1 scores. The values here are the best of 10 iterations for each cell. We notice that in a couple of cases, the F1 score improves. The highest percentage increase in F1 score is for Cora when we go from 90% $\rightarrow$ 10$\%$.

Dataset	Change in spectral moment		
	90% → 10%	90% → 50%	50% → 10%
Cora	(0.71,0.70) (0.80,0.77) (12.67%,10%)	(0.67,0.66) (0.68,0.67) (1.49%,1.51%)	(0.74,0.70) (0.80,0.78) (8.1%,11.4%)
Citeseer	(0.52,0.43) (0.56,0.48) (7.69%,11.62%)	(0.54,0.47) (0.58,0.50) (7.4%,6%)	(0.576,0.482) (0.570,0.482) (-1.04%,0.0%)
Pubmed	(0.745,0.693) (0.780,0.754) (4.69%,8.69%)	(0.73,0.67) (0.74,0.67) (1.36%,0%)	(0.771,0.751) (0.776,0.756) (0.67%,0.66%)
PPI	(0.25,0.08) (0.27,0.076) (8%,-5%)	(0.263,0.080) (0.274,0.076) (4.18%,-5%)	(0.271,0.081) (0.277,0.076) (2%,-6.17%)
Facebook	(0.60,0.592) (0.62,0.60) (3.33%,1.33%)	(0.54,0.50) (0.55,0.51) (1.85%,2%)	(0.65,0.62) (0.68,0.64) (4.61%,3.22%)

Table 4.1: Micro F1 and Macro F1 for node classification before and after graph augmentation. The row labels are the datasets used for testing the change in spectral moments. The column labels indicate changes in spectral moments. For example, $90 \rightarrow 10$ is augmenting a 90% subgraph to the 10% highest performing subgraph, where performance is measured using Micro-F1 score and Macro-F1 score. Each cell contains Micro-F1 score and Macro-F1 score before augmentation, Micro-F1 score and Macro-F1 score after augmentation, and percentage improvement in Micro-F1 score and Macro-F1 score, in that respective order.

4.4.2 Changes in Sparsity

Refer to Table 4.2 for changes in Micro-F1 and Macro-F1 scores before and after augmentation. As indicated in Section 4.4.1, the columns indicate the change in the spectral moment. In each cell, the first value is the number of edges before augmentation. The second value is the number of edges after augmentation. Finally, the third value is the percentage increase in the sparsity. The highest percentage increase, is for Citeseer, when we go from 90% $\rightarrow$ 10%. Additionally, we notice that the sparsity level improves in all cases.

Dataset	Change in spectral moment		
	90% $\rightarrow$ 10%	90% $\rightarrow$ 50%	50% $\rightarrow$ 10%
Cora	4243 843 80.1%	4152 3552 14.4%	1389 314 77.4%
Citeseer	3569 687 80.7%	3617 2093 42.1%	1117 299 73.2%
Pubmed	35468 15133 57.33%	34901 29007 16.88%	10346 5453 47.29%
PPI	634337 384316 39.4%	639937 574678 10.2%	197215 127040 35.58%
Facebook	136048 65633 51.75%	138715 97508 29.70%	42619 14684 65.54%

Table 4.2: Sparsity before and after graph augmentation. The row labels are the datasets used for testing the change in spectral moments. The column labels indicate changes in spectral moments. For example, 90 $\rightarrow$ 10 is augmenting a 90% subgraph to the 10% highest performing subgraph, where performance is measured using Micro-F1 score and Macro-F1 score. Each cell contains the number of edges in the graph before augmentation, the number of edges after augmentation, and the percentage increase in sparsity/drop in the number of edges.

4.4.3 Summary

In this section, we summarize the results of spectral augmentation. We use changes in node classification performance and sparsity to evaluate the performance of our augmentation strategy. Here Table 4.1 talks about improving node classification performance using the Micro-F1 score and Macro-F1 score. We notice that the node classification performance improves after augmentation, with the highest improvement observed for the Cora dataset when we augment a 90% subgraph's spectral moment to the best-performing 10% subgraph's spectral moment. In general, the least improvement observed was for the PPI dataset. Table 4.2 talks about the improvement in sparsity. The highest increase in sparsity is observed for the Citeseer dataset when we augment a 90% subgraph's spectral moment to the best-performing 10% subgraph's spectral moment. Overall sparsity increases for all the datasets shown here.

Chapter 5

Conclusion and Future Work

Representational learning focuses on learning embeddings for nodes, edges or the graph the graph itself to adequately capture some information regarding its structure. In the simplest terms possible, most representational learning models in graph focus on learning a mapping function, that maps nodes to real value embeddings. One approach to generating embeddings is using the concept of neural message passing. Message passing on a node, focuses on aggregating the embeddings of the node's neighbors and updating the node's embedding. Common methods that used message passing include methods such as, Graph Neural Networks(GNN) [7], Graph Convolutional Networks(GCN) [8], GraphSAGE [10] and, Graph Attention Networks(GAT) [9]. Graph augmentation focuses on the process of changing the structure of the graph to help improve classification performance and generalization to noisy data. Fundamentally there are 2 approaches to graph augmentation which include, *structure augmentation* and *feature augmentation*. From the definition commonly used in neural networks, a graph G can be written as (A,X), where A is the adjacency matrix and X is the feature matrix for the given graph. *Structure augmentation* focuses on bringing about changes to the adjacency matrix(A) of the graph [17] [18], and *feature augmentation* focuses on augmenting the feature matrix(X) of the given graph [12] [13]. The spectrum of a graph can be represented as the eigenvalues of the Normalized Laplacian of the graph($\lambda_1, \lambda_2, \lambda_3 \ldots \lambda_n$) and the l^{th} spectral moment of a graph can be represented as $m_l = \sum_{i=1}^{n} \lambda_i^l$

Our work focuses on producing graph augmentations using the spectral moments of a graph. According to the recent works of [1], the spectral moments of a graph have correlations to the structure of the graph, where the value of m_l is directly proportional to the average number of l-walks per node in the graph and inversely proportional to the average joint degree distribution of the l-walks in the graph. Keeping these properties in mind, we come up with a methodology, to augment the graph to a desired spectral moment. We also propose a method to identify the desired spectral moment that helps maximize accuracy and sparsity in a given graph. This is done by sampling subgraphs and interpolating surface plots colorizing the Micro-F1 score and Macro-F1 score. From this, we choose a common point that maximizes both. The node classification performance on the original graph and the augmented graph is verified using GraphSAGE. In the evaluation phase, we augment the graph from the spectral point from a 90 percent subgraph to that of a 10 percent subgraph(**90** $\rightarrow$ **10**), 90 percent subgraph to that of a 50 percent subgraph(**90** $\rightarrow$ **50**), and (**50** $\rightarrow$ **10**). According to Table4.1, we notice that in almost all cases, there is an improvement in the Miro-F1 score and Macro-F1 score, except for the PPI dataset where the Macro-F1 score dropped across all augmentation methods. We also notice that according to Table4.2, the sparsity of the graph goes up, when we augment to a spectral point of a smaller subgraph, in every case. Therefore, from our results, we attempt to give conclusive results that indicate the correlations between spectral moments and the model's performance.

Some future work in this area could include the following:

1. Learning-based methods that use spectral moments to augment the graph. Our method to augment a graph almost acts like a rule-based mechanism, that undergoes the same set of transformations for a given graph and destination moment. Furthermore, a learning-based mechanism can help decide how much augmentation is needed. For example, to increase the value of m_l, we remove the node with the highest degree and add independent l-cycles with that. These changes are static. To prevent this, the learning-based approach that can be more dynamic with the changes

being made, and thereby help converge at a more optimal solution.

2. Incorporating spectral moments into node embeddings can help create embeddings that can represent the structure of a node more clearly. Here one could use an approach very similar to STRUC2VEC [32], where a subgraph is sampled for each node. This sampled subgraph can be used to generate spectral moments representative of that node. Therefore, by using a learning process on these embeddings we can learn the structural similarity between the two nodes.

References

[1] S. Jin and R. Zafarani, “The spectral zoo of networks: Embedding and visualizing networks with spectral moments,” in *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery amp; Data Mining*, ser. KDD ’20. New York, NY, USA: Association for Computing Machinery, 2020, p. 1426–1434. [Online]. Available: https://doi.org/10.1145/3394486.3403195

[2] D. Wang, P. Cui, and W. Zhu, “Structural deep network embedding,” in *Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining*, 2016, pp. 1225–1234.

[3] Z. Zhang, L. Chen, F. Zhong, D. Wang, J. Jiang, S. Zhang, H. Jiang, M. Zheng, and X. Li, “Graph neural network approaches for drug-target interactions,” *Current Opinion in Structural Biology*, vol. 73, p. 102327, 2022. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0959440X2100169X

[4] P. Reiser, M. Neubert, A. Eberhard, L. Torresi, C. Zhou, C. Shao, H. Metni, C. van Hoesel, H. Schopmans, T. Sommer *et al.*, “Graph neural networks for materials science and chemistry,” *Communications Materials*, vol. 3, no. 1, p. 93, 2022.

[5] S. Vashishth, S. Sanyal, V. Nitin, and P. Talukdar, “Composition-based multi-relational graph convolutional networks,” *arXiv preprint arXiv:1911.03082*, 2019.

[6] X. Wang and A. Gupta, “Videos as space-time region graphs,” in *Proceedings of the European conference on computer vision (ECCV)*, 2018, pp. 399–417.

[7] F. Scarselli, M. Gori, A. C. Tsoi, M. Hagenbuchner, and G. Monfardini, “The graph neural network model,” *IEEE transactions on neural networks*, vol. 20, no. 1, pp.

61–80, 2008.

[8] T. N. Kipf and M. Welling, “Semi-supervised classification with graph convolutional networks,” *arXiv preprint arXiv:1609.02907*, 2016.

[9] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Lio, and Y. Bengio, “Graph attention networks,” *arXiv preprint arXiv:1710.10903*, 2017.

[10] W. Hamilton, Z. Ying, and J. Leskovec, “Inductive representation learning on large graphs,” *Advances in neural information processing systems*, vol. 30, 2017.

[11] T. Zhao, Y. Liu, L. Neves, O. Woodford, M. Jiang, and N. Shah, “Data augmentation for graph neural networks,” in *Proceedings of the aaai conference on artificial intelligence*, vol. 35, no. 12, 2021, pp. 11 015–11 023.

[12] P. Velickovic, W. Fedus, W. L. Hamilton, P. Liò, Y. Bengio, and R. D. Hjelm, “Deep graph infomax.” *ICLR (Poster)*, vol. 2, no. 3, p. 4, 2019.

[13] L. Yang, L. Zhang, and W. Yang, “Graph adversarial self-supervised learning,” *Advances in Neural Information Processing Systems*, vol. 34, pp. 14 887–14 899, 2021.

[14] Y. You, T. Chen, Y. Sui, T. Chen, Z. Wang, and Y. Shen, “Graph contrastive learning with augmentations,” *Advances in neural information processing systems*, vol. 33, pp. 5812–5823, 2020.

[15] L. Yang, Z. Kang, X. Cao, D. Jin, B. Yang, and Y. Guo, “Topology optimization based graph convolutional network.” in *IJCAI*, 2019, pp. 4054–4061.

[16] K. Ding, Z. Xu, H. Tong, and H. Liu, “Data augmentation for deep graph learning: A survey,” *ACM SIGKDD Explorations Newsletter*, vol. 24, no. 2, pp. 61–77, 2022.

[17] Y. Rong, W. Huang, T. Xu, and J. Huang, "Dropedge: Towards deep graph convolutional networks on node classification," *arXiv preprint arXiv:1907.10903*, 2019.

[18] J. Guo, L. Du, W. Bi, Q. Fu, X. Ma, X. Chen, S. Han, D. Zhang, and Y. Zhang, "Homophily-oriented heterogeneous graph rewiring," *arXiv preprint arXiv:2302.06299*, 2023.

[19] J. Topping, F. Di Giovanni, B. P. Chamberlain, X. Dong, and M. M. Bronstein, "Understanding over-squashing and bottlenecks on graphs via curvature," in *International Conference on Learning Representations*.

[20] Y. Jiao, Y. Xiong, J. Zhang, Y. Zhang, T. Zhang, and Y. Zhu, "Sub-graph contrast for scalable self-supervised graph representation learning," in *2020 IEEE international conference on data mining (ICDM)*. IEEE, 2020, pp. 222–231.

[21] H. Park, S. Lee, S. Kim, J. Park, J. Jeong, K.-M. Kim, J.-W. Ha, and H. J. Kim, "Metropolis-hastings data augmentation for graph neural networks," *Advances in Neural Information Processing Systems*, vol. 34, pp. 19 010–19 020, 2021.

[22] J. Gilmer, S. S. Schoenholz, P. F. Riley, O. Vinyals, and G. E. Dahl, "Neural message passing for quantum chemistry," in *International conference on machine learning*. PMLR, 2017, pp. 1263–1272.

[23] M. Belkin and P. Niyogi, "Laplacian eigenmaps and spectral techniques for embedding and clustering," *Advances in neural information processing systems*, vol. 14, 2001.

[24] A. Ahmed, N. Shervashidze, S. Narayanamurthy, V. Josifovski, and A. J. Smola, "Distributed large-scale natural graph factorization," in *Proceedings of the 22nd international conference on World Wide Web*, 2013, pp. 37–48.

[25] S. Cao, W. Lu, and Q. Xu, “Grarep: Learning graph representations with global structural information,” in *Proceedings of the 24th ACM international on conference on information and knowledge management*, 2015, pp. 891–900.

[26] M. Ou, P. Cui, J. Pei, Z. Zhang, and W. Zhu, “Asymmetric transitivity preserving graph embedding,” in *Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining*, 2016, pp. 1105–1114.

[27] B. Perozzi, R. Al-Rfou, and S. Skiena, “Deepwalk: Online learning of social representations,” in *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, 2014, pp. 701–710.

[28] A. Grover and J. Leskovec, “node2vec: Scalable feature learning for networks,” in *Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining*, 2016, pp. 855–864.

[29] J. Tang, M. Qu, M. Wang, M. Zhang, J. Yan, and Q. Mei, “Line: Large-scale information network embedding,” in *Proceedings of the 24th international conference on world wide web*, 2015, pp. 1067–1077.

[30] T. Mikolov, K. Chen, G. Corrado, and J. Dean, “Efficient estimation of word representations in vector space,” *arXiv preprint arXiv:1301.3781*, 2013.

[31] B. Perozzi, V. Kulkarni, H. Chen, and S. Skiena, “Don’t walk, skip! online learning of multi-scale network embeddings,” in *Proceedings of the 2017 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining 2017*, 2017, pp. 258–265.

[32] L. F. Ribeiro, P. H. Saverese, and D. R. Figueiredo, “struc2vec: Learning node representations from structural identity,” in *Proceedings of the 23rd ACM SIGKDD international conference on knowledge discovery and data mining*, 2017, pp. 385–394.

[33] A. Graves and A. Graves, "Long short-term memory," *Supervised sequence labelling with recurrent neural networks*, pp. 37–45, 2012.

[34] K. Xu, W. Hu, J. Leskovec, and S. Jegelka, "How powerful are graph neural networks?" *arXiv preprint arXiv:1810.00826*, 2018.

[35] B. Weisfeiler and A. Leman, "A reduction of a graph to a canonical form and an algebra arising during this reduction, nauchno–technicheskaja informatsia, 9 (1968), 12–16."

[36] Y. You, T. Chen, Y. Shen, and Z. Wang, "Graph contrastive learning automated," in *International Conference on Machine Learning*. PMLR, 2021, pp. 12 121–12 132.

[37] Y. Wang, W. Wang, Y. Liang, Y. Cai, J. Liu, and B. Hooi, "Nodeaug: Semi-supervised node classification with data augmentation," in *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, 2020, pp. 207–217.

[38] V. Verma, A. Lamb, C. Beckham, A. Najafi, I. Mitliagkas, D. Lopez-Paz, and Y. Bengio, "Manifold mixup: Better representations by interpolating hidden states," in *International conference on machine learning*. PMLR, 2019, pp. 6438–6447.

[39] W. Bi, L. Du, Q. Fu, Y. Wang, S. Han, and D. Zhang, "Make heterophily graphs better fit gnn: A graph rewiring approach," *arXiv preprint arXiv:2209.08264*, 2022.

[40] C. Zheng, B. Zong, W. Cheng, D. Song, J. Ni, W. Yu, H. Chen, and W. Wang, "Robust graph representation learning via neural sparsification," in *Proceedings of the 37th International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, H. D. III and A. Singh, Eds., vol. 119. PMLR, 13–18 Jul 2020, pp. 11 458–11 468. [Online]. Available: https://proceedings.mlr.press/v119/zheng20d.html

[41] B. Hui, D. Yan, X. Ma, and W.-S. Ku, “Rethinking graph lottery tickets: Graph sparsity matters,” in *International Conference on Learning Representations (ICLR)*, 2023.

[42] J. Qiu, Q. Chen, Y. Dong, J. Zhang, H. Yang, M. Ding, K. Wang, and J. Tang, “Gcc: Graph contrastive coding for graph neural network pre-training,” in *Proceedings of the 26th ACM SIGKDD international conference on knowledge discovery & data mining*, 2020, pp. 1150–1160.

[43] K. Xu, H. Chen, S. Liu, P.-Y. Chen, T.-W. Weng, M. Hong, and X. Lin, “Topology attack and defense for graph neural networks: An optimization perspective,” *arXiv preprint arXiv:1906.04214*, 2019.

[44] X. Zhang and M. Zitnik, “Gnnguard: Defending graph neural networks against adversarial attacks,” *Advances in neural information processing systems*, vol. 33, pp. 9263–9275, 2020.

[45] H. Wu, C. Wang, Y. Tyshetskiy, A. Docherty, K. Lu, and L. Zhu, “Adversarial examples on graph data: Deep insights into attack and defense,” *arXiv preprint arXiv:1903.01610*, 2019.

[46] I. Amidror, “Scattered data interpolation methods for electronic imaging systems: a survey,” *Journal of electronic imaging*, vol. 11, no. 2, pp. 157–176, 2002.

[47] K. Paton, “An algorithm for finding a fundamental set of cycles of a graph,” *Communications of the ACM*, vol. 12, no. 9, pp. 514–518, 1969.

[48] A. K. McCallum, K. Nigam, J. Rennie, and K. Seymore, “Automating the construction of internet portals with machine learning,” *Information Retrieval*, vol. 3, pp. 127–163, 2000.

[49] C. L. Giles, K. D. Bollacker, and S. Lawrence, “Citeseer: An automatic citation indexing system,” in *Proceedings of the Third ACM Conference on Digital Libraries*, ser. DL ’98. New York, NY, USA: Association for Computing Machinery, 1998, p. 89–98. [Online]. Available: https://doi.org/10.1145/276675.276685

[50] P. Sen, G. Namata, M. Bilgic, L. Getoor, B. Galligher, and T. Eliassi-Rad, “Collective classification in network data,” *AI magazine*, vol. 29, no. 3, pp. 93–93, 2008.

[51] B. Rozemberczki, C. Allen, and R. Sarkar, “Multi-Scale Attributed Node Embedding,” *Journal of Complex Networks*, vol. 9, no. 2, 2021.

[52] D. Cohen-Steiner, W. Kong, C. Sohler, and G. Valiant, “Approximating the spectrum of a graph,” in *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery amp; Data Mining*, ser. KDD ’18. New York, NY, USA: Association for Computing Machinery, 2018, p. 1263–1271. [Online]. Available: https://doi.org/10.1145/3219819.3220119

Atul Anand Gopalakrishnan

Email ID: reachme.atul@gmail.com | **Ph. No:** (315)-450-9127

EDUCATION

Syracuse University - College of Engineering & Computer Science, Syracuse, NY August 2021 - May 2023
M.S. in Computer & Information Sciences **GPA: 3.917/4**
PES University - Department of Computer Science, Bangalore, Karnataka June 2017 – July 2021
BTech in Computer Science **GPA: 8.50/10**

LANGUAGES AND FRAMEWORKS:

C, C++, Java, Javascript, Python, Haskell, Apache Hadoop, Apache Spark, Apache Kafka, Apache Storm, Microservices, Docker, Flask, AWS, Tensorflow and Keras, OpenMP, OpenCL

EXPERIENCE

Data Science Intern, Roambee, California, USA Jun 2022-Sept 2022
I worked with the data science team to create APIs that estimated sensor requirements at different granularities, including customer and geographical levels. Accuracy(Mean Absolute Error) was +/-26.83 across all the organizations.

PROJECT

Detecting hateful memes using knowledge graphs Sept 2022-Present

- Created a hateful meme detector with the help of knowledge graphs. Done on the Hateful Memes Dataset published by Meta
- Extracted image and text captions and mapped the entities to their respective Wikipedia page.
- Generated knowledge triplets for the Wikipedia page using the Partition Filter Network and constructed a knowledge graph using these triplets
- Obtained embeddings for the entities and relations using knowledge graph embedding models and classified the same using an AdaBoost Classifier. Obtained an accuracy of 54.51 percent with benchmark accuracy being 69.7 percent.
- **NOTE:** Since this is an ongoing project, some of the details are still underway.

MS Thesis, Graph Representation Learning Jan 2022-Present

- Working on my thesis in the domain of graph representation learning under the supervision of Prof. Reza Zafarani.
- At the current stage, I have built my fundamentals on graph representation learning by reading the Deep Learning on Graphs book, alongside some recent research work across major conferences like NeurIPS, SIGKDD, ICLR, ICML, and some more, and ideating with my guide on a regular basis.
- Our current work focuses on estimating the classification accuracies for embedding models using some spectral graph properties.
- **NOTE:** Since this is an ongoing project, some of the details are still underway

HACS: Access Control for Streaming Data Across Heterogeneous Communication Models July 2020 - April 2021

- Created and devised uniform access control mechanisms across 2 big-data communication models, namely the producer-consumer model and point-to-point model Apache Kafka and Apache Storm with Java
- Developed by punctuating each message with respective access control using the concept of security punctuations. The access control is embedded on the Producer side in Kafka and imposed at the Bolt in Storm
- Accepted for a short paper at the World AI and IoT Congress(AIIoT), 2020.

PTangle: A Parallel Detector for Unverified Blockchain Transactions January 2020 - July 2020

- Parallelized Random Weighted Walks on a Directed Acyclic Graph-based Blockchain to implement parallel tip selection and profiled same using PyMP and Numba
- Produced a peak speedup of 73% for a large number of transactions and near 50% speedup for a small number of transactions
- Accepted as a short paper at the International Conference on Algorithms & Architectures for Parallel Processing(ICA3PP) 2020.

AWARDS AND CERTIFICATIONS

All India 2nd place in e-Yantra Robotics Competition hosted by IIT Bombay in 2020. Built a service drone using concepts of Robotic Operating systems, Image processing, and Scheduling so the system performs optimally