HIERARCHICAL GRAPH NEURAL NETWORK FOR LEARNING HIGH-DIMENSIONAL GENE REGULATORY NETWORKS

Presented in Partial Fulfillment of
the Requirements for the Degree of

MASTER OF SCIENCE

with a Major in

Statistical Science

in the

College of Graduate Studies

University of Idaho

by

ELIJAH DANQUAH DARKO

Major Professor
AUDREY QIUYAN FU, PH.D.

Committee
CHRISTOPHER WILLIAMS, PH.D.
BENJAMIN RIDENHOUR, PH.D.

Department Administrator
HIROTACHI ABO, PH.D.

MAY 2023

ABSTRACT

The study of Gene Regulatory Networks (GRNs) poses a significant challenge due to their high dimensions, making it difficult to accurately infer these networks for many genes. Moreover, it is also a challenge to interpret these networks or generate biologically meaningful hypotheses for further investigation. To address this issue, researchers often look for modules and pathways, which are subnetworks of much smaller sizes. This approach suggests an alternative to studying GRNs, which is to learn about a hierarchy of multiple layers of networks.

The idea behind this approach is to use a hierarchy of networks, with each layer building on the one below it. In this hierarchy, a node in an upper-layer network corresponds to a subnetwork in the lower-layer network. Therefore, the size of the networks in each layer increases from top to bottom in the hierarchy. This means that the bottom-layer network is the GRN at the gene level, whereas upper-layer networks group genes into subnetworks or networks of these subnetworks. This reduces the dimensions(number of nodes) as we move up the hierarchy and allows us to "zoom out" and look at the big picture. While this approach is similar to hierarchical clustering, it differs in that we are interested not only in the hierarchy but also in learning about the relationships among clusters at each level.

To address the challenges of inferring hierarchical GRNs from genome-wide data of molecular phenotypes (such as gene expression, DNA methylation, etc.), we present a deep learning algorithm, Hierarchical Graph Neural Network (HGNN). Our HGNN algorithm updates the inference on the hierarchy over iterations, in addition to updating the inference of the network in each layer. We utilize a four-component loss function that aims to recover the underlying network structure.

To assess the performance of this algorithm, we perform extensive simulations on two main datasets generated from a disconnected two-layer hierarchy and a connected three-layer hierarchy. By using HGNN, we aim to accurately infer hierarchical GRNs, allowing us to better understand the underlying biological processes and generate hypotheses for further

investigation. Additionally, based on the results, HGNN has the potential to be applied to other datasets with high dimensions, offering a promising solution for studying complex biological systems.

ACKNOWLEDGMENTS

I would like to extend my heartfelt thanks to all those who have contributed to the successful completion of this thesis.

Firstly, I am immensely grateful to my supervisor, Dr. Fu, for their invaluable guidance, support, and motivation throughout my research journey. Their expertise, insightful feedback, and patience have been crucial in shaping my ideas and improving the quality of my work.

I am also deeply appreciative of Dr. Benjamin and Dr. Chris, who generously shared their time and expertise as members of my thesis committee. Their constructive feedback, observations, and rigorous critique have pushed me to think critically and deeply about my research topic.

My gratitude also goes to the University of Idaho Department of Mathematics and Statistical Science for providing a dynamic academic environment and access to cutting-edge facilities and equipment.

I am grateful to my colleagues and friends in Dr. Fu's lab for their valuable feedback, encouragement, and support, which helped me refine my research ideas and strengthen my arguments. I especially appreciate Jarred Kvamme for his help with the code implementation.

Finally, I would like to express my heartfelt appreciation to my family for their unwavering love, support, and encouragement. Their belief in me has been a constant source of motivation and inspiration, and I am truly grateful for their presence in my life.

In conclusion, I am deeply grateful to everyone who has played a role in my research journey. Your support, encouragement, and guidance have been crucial in helping me achieve my academic goals, and I am truly thankful for everything you have done for me.

DEDICATION

I dedicate this work to my parents and family for their love and support.

TABLE OF CONTENTS

List of Tables

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

In this chapter, we introduce Graph Neural Networks (GNNs) and highlight their applications, Gene Regulatory Networks (GRNs), and the use of GNNs in inference on GRNs.

1.1 GRAPH NEURAL NETWORKS

Graph Neural Networks (GNNs) are a specific type of neural network designed to analyze and process inputs that can be graph-structured. The generalization of Convolutional Neural Networks (CNNs)(Hinton and Salakhutdinov, 2006) to unusual graph structures is how GNNs are commonly defined (Reiser *et al.*, 2022).

These networks can learn representations of graph-structured data by including the topological details of the graph structure in the network design. GNNs' ability to learn from graph-structured data makes them very appealing for a variety of applications, such as social network analysis, bioinformatics, and recommendation systems, among others.

The history of GNNs is dated to the early 2000s (Scarselli *et al.*, 2008) when researchers first started looking into the application of neural networks to the processing of graph-structured data. Graph kernels are functions that map a pair of graphs to a real value, measuring the similarity between the graphs (Vishwanathan *et al.*, 2010; Ghosh *et al.*, 2018). These techniques were developed on the basis that machine learning algorithms might use properties derived from graph-theoretical approaches to learn some hidden relationships in the network. However, these methods were computationally expensive and limited in their ability to handle large, complex graphs (Vishwanathan *et al.*, 2010; Kriege *et al.*, 2020).

The success of CNNs in image and signal processing has led to a resurgence in interest in GNNs in recent years. Researchers developed the idea of Graph Convolution Networks (GCNs) and began investigating the usage of CNNs for processing graph-structured data.

Introduced by Kipf and Welling (2016), a GCN is a type of GNN that learns representations of graph-structured data using convolutional functions and operations. Numerous applications, such as node classification, link prediction, and graph classification, have demonstrated the efficacy of GCNs.

GNNs are built on the concept of information transmission, in which the representation or signals of nodes are updated by combining those of their neighbors. Additionally, GNNs can capture the local neighborhood information of each node while maintaining the global structure of the network. This procedure is performed several times, and the final representations are applied to the relevant task (node classification, link prediction, etc.). Since the introduction of GCNs, there have been many other types of GNNs proposed and developed, each with their own unique characteristics and capabilities. Some examples of these include:

- Graph Attention Networks (GATs): First proposed by Veličković *et al.* (2017), these networks use attention mechanisms to weigh the importance of different nodes in the graph during the convolutional operations.

- Graph Recurrent Networks: These networks handle graph-structured data by repeatedly updating the node representations using Recurrent Neural Networks (RNNs). This method enables the temporal dynamics of the graph data to be captured (Mikolov *et al.*, 2010).

- Graph Autoencoder Networks (GAEs): These networks learn low-dimensional representations of graph-structured data using autoencoder architectures. To perform tasks like link prediction or graph construction, GAEs aim to learn a concise representation of the graph structure (Pan *et al.*, 2018).

GNNs have demonstrated their effectiveness in applications and are a robust tool for analyzing graph-structured data. New training methods and architectures frequently proposed and created in the domain of research, actively involve the development of GNNs.

Though GNNs have yielded promising outcomes, it is important to note that the currently available GNNs also have significant drawbacks and difficulties. For instance, the interpretability of GNNs is not well understood, and there may be challenges in scaling GNNs to handle very large graphs. To enhance their functionality and applicability for use in solving real-world issues, there is therefore still a great deal of room for further study in GNNs.

Traditional approaches for inferring GRNs rely on correlation-based or mutual information-based techniques, which can be noise-sensitive and have limited capacity to handle non-linear interactions. On the other hand, GNNs may be trained using information about gene expression and previously observed interactions between genes to forecast newly discovered gene interactions, which can provide a new understanding of the genes and their interactions.

GNNs have been used in a variety of domains, including computer vision, chemistry, natural language processing, physics, social network analysis, bioinformatics, and recommendation systems among others.

- On graph-structured text data, such as social networks (Facebook, LinkedIn, and Twitter), road networks (used in navigation tools like GPS, Google Maps, and Apple Maps), and document collections, GNNs have been utilized in text categorization and sentiment analysis in natural language processing.

- GNNs have been utilized in chemistry to predict the characteristics of molecules and materials, including medicinal effectiveness and toxicity (Reiser *et al.*, 2022).

- For object recognition and segmentation of graph-structured pictures, such as road networks and 3D point clouds, GNNs have been employed in computer vision (Shi and Rajkumar, 2020).

- GNNs have been applied to physics to explore the dynamics of graph-based physical systems, such as quantum systems and protein folding.

- GNNs have also been widely used in bioinformatics and biological sciences (Wang *et al.*, 2021; Zitnik *et al.*, 2018).

1.2 GENE REGULATORY NETWORKS (GRNS)

One of the foundational concepts in biology is that genes do not function independently of each other. However, the development of technique/methods for inferring regulatory networks from gene expression data has been a gradual process. This presents a challenge to understanding the hidden interactions between genes and their regulation in biological systems (Chen and Mar, 2018). Defined by Emmert-Streib *et al.* (2014), GRNs are networks that has been inferred from gene expression data obtained through techniques such as ribonucleic acid(RNA) sequencing, deoxyribonucleic acid(DNA) microarrays among others (Singh *et al.*, 2018), and they they can be highly dense and sparse making it difficult to draw any biological hypothesis. However, interest in GRNs has increased in recent years as a result of its ability to offer potential insights into complex biological processes, help identify genes and pathways involved in disease development, and the development of new therapeutic strategies. The inference of GRNs can be performed at different levels of resolution, from whole-network to sub-level. Thus, researchers have explored the use of both classical and machine/deep learning methods to develop algorithms for GRN inference. Some of these include partial correlation, bayesian networks, context likelihood relatedness, scGNN, ARACNE, and GENIE3, among others (Schafer and Strimmer, 2005; Wang *et al.*, 2021; Friedman *et al.*, 2000; Faith *et al.*, 2007; Krouk *et al.*, 2013; Allen *et al.*, 2012; Badsha *et al.*, 2019; Huynh-Thu *et al.*, 2010; Reiss *et al.*, 2016; Kipf and Welling, 2017).

In this work, we present a deep learning algorithm , Hierarchical Graph Neural Network (HGNN) for learning high-dimensional GRNs. In upcoming chapters, we will elaborate on the methodology that we have used in this work, then discuss our simulation study, present our results, and perform a thorough analysis of our findings. Finally, in the last chapter, we

will provide a conclusion and discussion of our study, highlighting the implications of our results, and the potential areas for future research.

CHAPTER 2

METHODOLOGY

In this chapter, we first give a brief introduction to some fundamental concepts in graph theory, which serve as a foundation for discussing our Hierarchical Graph Neural Network (HGNN) architecture and its various components.

2.1 CONCEPTS IN GRAPH THEORY

A graph G consists of a set of vertices $V(G)$ and a set of edges, $E(G)$. Edges (or links) represent connections between vertices. Two vertices in a graph are said to be adjacent if there is an edge joining them. Adjacent vertices are said to be incident to the edge joining them. For a vertex v in the graph G, we call the set of vertices in G that are adjacent to it as its open neighborhood. We will let $N_G(v)$ denote the open neighborhood of a vertex v in graph G. The cardinality of $N_G(v)$ is called the degree of v, denoted by $\deg(v)$. It is easy to infer that $\deg(v)$ is also the number of edges that are incident to the vertex v.

The graph G is directed if its edges have a direction and contain a cycle if we can move from a vertex back to the same vertex, without repeating edges and vertices. G is called acyclic if it does not contain any cycle. We define a graph G_d as directed and acyclic. G_d is called a directed acyclic graph (DAG). DAGs are useful graph structures used in modeling graph connectivity, probabilities, and casualty, among others. In this work, we make use of DAGs in our simulation study to simulate hierarchies from which we generate a pseudo gene expression data. In modeling GRNs using graph-based approaches, each gene is represented as a node in the graph, and the regulatory interactions between genes are represented as edges in the graph. These interactions can be either direct or indirect, and they can either activate or repress the expression of the genes.

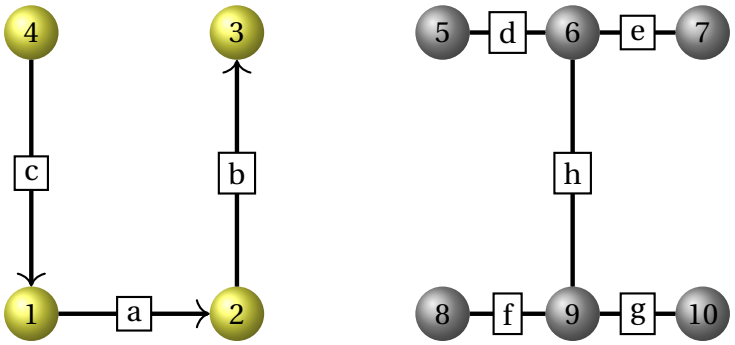


FIGURE 2.1: A graph G with two subgraphs identified by the color of nodes with node and edge set are given by $V(G) = \{1,2,3,4,5,6,7,8,9,10\}$ and $E(G) = \{a,b,c,d,e,f,g,h\}$ respectively. Nodes labeled 5 and 6 are adjacent since they are connected by the edge d. Additionally, nodes 5 and 6 are incident to the edge d. We also have $N_G(6) = \{5,7,9\}$, hence deg(6) = 3. Here, the subgraph with yellow-colored nodes is a directed graph.

2.1.1 *Adjacency Matrix*

Let G be a graph with n number of vertices. The adjacency matrix $A(G)$ of G is an $n \times n$ matrix whose entries reflect the connection or relationship between the vertices in G. If the edges of G have no embedded weights, the $ij-$ entries of its adjacency matrix are defined as,

$$a_{ij} = \begin{cases} 1 & \text{if } i,j \in V(G) \text{ and } (i,j) \in E(G)\,, \\ 0 & \text{otherwise.} \end{cases}$$

By definition, if G is undirected, then $A(G)$ is a symmetric matrix (i.e., $a_{ij} = a_{ji}$). Each row or column's total is also equal to the degree of the vertex it represents. As a result, we can deduce that the total degree of the vertices equals the total of all the entries in $A(G)$. We also note that the number of ones in the upper or lower triangular matrix is equal to the number of edges in G. $A(G)$ has its diagonal entries being zeros(acyclic), hence the trace of $A(G)$ is zero. As a result, its eigenvalues add up to 0.

2.1.2 *Degree Matrix*

The degree matrix $\mathscr{D}(G)$ of a graph G is an $n \times n$ diagonal matrix whose entries are the degree of the vertices of G. Mathematically, this can be represented as

$$d_{ii} = \begin{cases} \deg(v_i) & \text{where } v_i \text{ represent the } i^{th} \text{ vertex of } G \\ 0 & \text{otherwise.} \end{cases}$$

2.1.3 *Laplacian Matrix*

The laplacian matrix $\mathscr{L}(G)$ of a graph G is defined as

$$\mathscr{L}(G) = \mathscr{D}(G) - A(G)$$

where $A(G)$ and $\mathscr{D}(G)$ represent the adjacency and degree of matrix of the graph, G respectively. We also defined the signless laplacian, $\mathscr{L}_s(G)$ and normalized laplacian $\mathscr{L}_n(G)$ matrices, of G. These are given by

$$\mathscr{L}_s(G) = \mathscr{D}(G) + A(G),$$

and

$$\mathscr{L}_n(G) = \mathscr{D}(G)^{-1/2} \left[\mathscr{D}(G) - A(G)\right] \mathscr{D}(G)^{-1/2}.$$

The spectrum of G, $\gamma(G)$ is the set of all eigenvalues of its matrix representation. A graph's set of eigenvalues and eigenvectors conveys important information about its structure.
Taking graph G in Figure 2.1 we define its matrix representations below;
The adjacency matrix is given by

$$A(G) = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \\ 7 \\ 8 \\ 9 \\ 10 \end{array} \begin{array}{c} \begin{array}{cccccccccc} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 \end{array} \\ \left(\begin{array}{cccc|cccccc} 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ \hline 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{array}\right) \end{array}$$

It can be seen that the off diagonals of the partitions $A(G)$ are a submatrix of zeros, indicating the disconnection between the two subgraphs. Additionally, the first diagonal submatrix is asymmetric (from the directed subgraph), with the second being a symmetric submatrix (from the undirected subgraph).

The degree matrix given as;

$$\mathscr{D}(G) = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \\ 7 \\ 8 \\ 9 \\ 10 \end{array} \begin{array}{c} \begin{array}{cccccccccc} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 \end{array} \\ \begin{pmatrix} 2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 3 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 3 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix} \end{array}$$

By definition, the signless laplacian matrix is given by,

$$\mathscr{L}_s(G) = A(G) + \mathscr{D}(G)$$

$$= \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ \hline 5 \\ 6 \\ 7 \\ 8 \\ 9 \\ 10 \end{array} \begin{array}{c} \begin{array}{cccc|cccccc} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 \end{array} \\ \left(\begin{array}{cccc|cccccc} 2 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 2 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ \hline 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 3 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 3 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \end{array}\right) \end{array}$$

2.1.4 *Topological Order*

Let G_D be a DAG. The topological order of G_D is an ordering in which the nodes of G_D are linearly arranged. This means for any directed edge (a,b), the node a appears before the node b in the linear arrangement(Haeupler *et al.*, 2012). In other words, it is the complete ordering of the graph's nodes such that each directed edge leads from an earlier node to a later node.

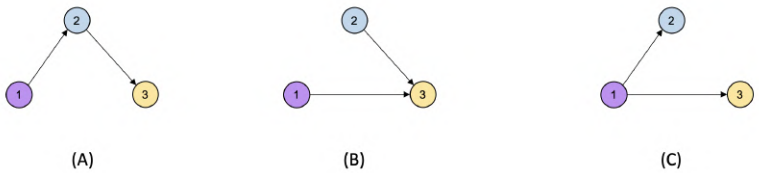


FIGURE 2.2: Directed acyclic graphs. The topological ordering of these DAGs are given by $\{1,2,3\}$, $\{1,2,3\}$ and $\{1,3,2\}$ respectively.

2.2 HIERARCHICAL GRAPH NEURAL NETWORK (HGNN)

In this section, we describe our proposed deep learning algorithm, Hierarchical Graph Neural Network (HGNN), and its components, for learning a highly dimensional GRNs.

2.2.1 *The Algorithm*

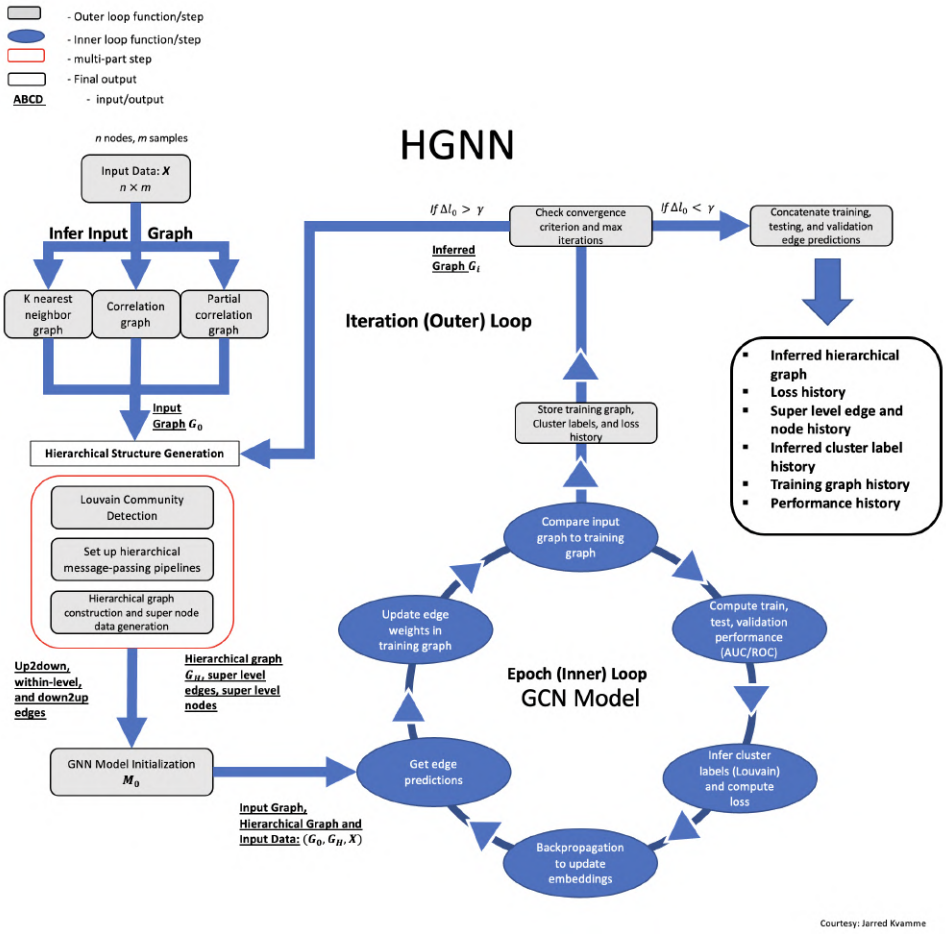


FIGURE 2.3: The architecture of Hierarchical Graph Neural Networks. At the i^{th} iteration, the algorithm utilizes gene expression data to infer an input graph and generate a hierarchical structure with pipelines for message or signal transfer. This structure is then fed into the Graph Convolution Network model, resulting in an inferred graph that serves as the input graph for the $(i+1)^{th}$ iteration until the change in loss meets a predefined criterion. This iterative process aims to produce a more accurate inference of the underlying structure in the network.

We can emphasize four main components of the overall architecture of our method, HGNN, as shown in Figure 2.3.

2.2.1.1 *Graph Construction*

The gene expression value for an individual and a specific gene is a quantitative measure of the level of activity of that gene in the cells of that individual. These measurements yields scalar values which can expressed as a matrix of dimension $n \times m$, where n is the number of genes (nodes) and m, is the number of samples or individuals. Let $\mathscr{R}$ be the correlation matrix representing the linear relationship or dependence between genes of dimension $n \times n$. $\mathscr{R}$ is a symmetric matrix, with its $ij-$ entries being the pairwise correlation between genes. This can be defined as

$$
\begin{aligned}
a_{ij} &= r \\
&= \frac{\sum_{i=1}^{m}(x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum_{i=1}^{m}(x_i - \bar{x})^2 \times \sum_{i=1}^{m}(y_i - \bar{y})^2}} \\
&= \frac{m\sum_{i=1}^{m} x_i y_i - \left(\sum_{i=1}^{m} x\right)\left(\sum_{i=1}^{m} y\right)}{\sqrt{\left[m\sum_{i=1}^{m} x_i^2 - \left(\sum_{i=1}^{m} x_i\right)^2\right]\left[m\sum_{i=1}^{m} y_i^2 - \left(\sum_{i=1}^{m} y_i\right)^2\right]}}
\end{aligned}
$$

where $\bar{x}, \bar{y}$ represents the sample mean for $i = 1, \ldots, m$. Generating the correlation matrix serves as the initial stage for our graph construction process. To get the graph adjacency matrix, A_c which leads us to the final stage of the graph construction process. The entries of A_c are defined as,

$$
a_{ij} = \begin{cases} 0 & \forall\ i = j \\ 1 & \text{if } |r| \geq c, \forall\ i \neq j \\ 0 & \text{otherwise.} \end{cases}
$$

where $c \in [0,1]$ is a cutoff or threshold for the pairwise correlations between genes. The value of c can randomly chosen based of some known underlying relationship between the genes. $|r| \in [0,1]$, the correlation coefficient. It can be observed that the diagonal entries of A_c become zeros. This is to avoid the occurrence of self-loops when generating a graph from A_c. The final stage of the graph construction process is to generate an initial undirected network, G_0 from A_c. This serves as an input graph to our HGNN algorithm for the initial start iteration.

2.2.1.2 *Hierarchical Structure Generation*

Hierarchical Structure Generation is a key component of HGNN, which comprises a Louvain community detection algorithm and a message propagation process. Introduced by Blondel *et al.* (2008), the Louvain algorithm has gained popularity in finding modular structures or clusters from complex network structures using two main iterative phases or steps.

Let G_0 be the initial graph input generated from the correlation adjacency matrix. In the first phase (bottom-up phase) of the Louvain algorithm on G_0, each node in G_0 is assigned its own community. Thus, at this stage, the total number of communities is equal to the number of nodes in G_0. Then, considering the neighbors j of each node i, the algorithm assesses the change in modularity (that is, a measure of the strength within modules or communities) that would result from moving the node i from its community to the community of its neighbor j. Once this process is applied to all neighbors j of the given node i, the node is assigned to the community where the highest positive modularity gain is achieved. This process is repeated in succession for all nodes in G_0 until there is no modularity better change.

The second phase(top-down phase) of the algorithm involves an aggregation process and the first phase. Here, each community from the first phase is considered a new network or node, with its edge information obtained by an aggregation of edges from the communities obtained in the first phase. Phase one is repeated to obtain a new community structure.

Using communities as nodes, this phase is iterated, until no further modularity improvement is possible.

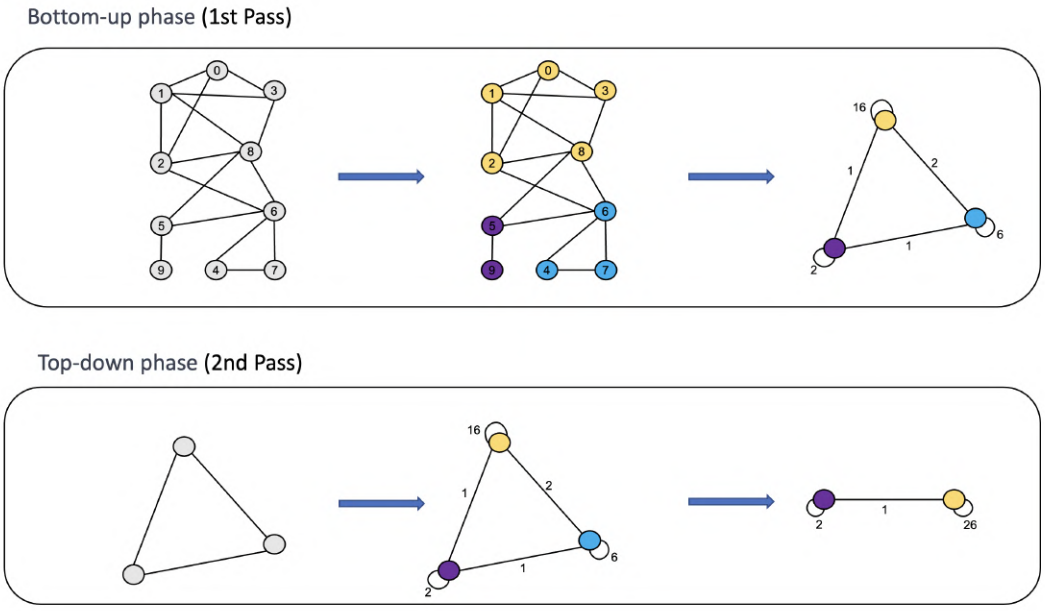


FIGURE 2.4: Visualization of the steps in the Louvain algorithm. The first pass is the first stage of modularity optimization to find modules. The second phase takes modules in the first as nodes and finds module of modules by optimizing modularity. Passes are iteratively repeated until mode shifting does not provide possible modularity improvement. Self-loop edges represent twice the edges in a module or cluster and cross edges represent edges existing between modules or clusters.

The next stage after generating a hierarchy from the Louvain method is building pipelines for message or signal transfer. HGNN adopts three message propagation techniques from Zhong *et al.* (2023). These are;

- Down-up propagation. This technique updates the node representation of an upper layer in the hierarchy using the previous layer by calculating the weighted sum of node representation. This sends information from lower layers to the higher layer in the hierarchy allowing the capturing of both local and global information from the hierarchy.

- Within-level propagation. In within-level propagation, node representations are updated by aggregating neighborhood information together with a node's information at the same level in the hierarchy.
- Top-down propagation. In this technique, node representations in higher layers of are used to update the representations of nodes in the initial bottom layer. This is done using the information obtained during the down-up propagation step.

  of the initial bottom layer using the representations obtained from the down-up propagation step.

2.2.1.3 *GCN with e number of epochs*

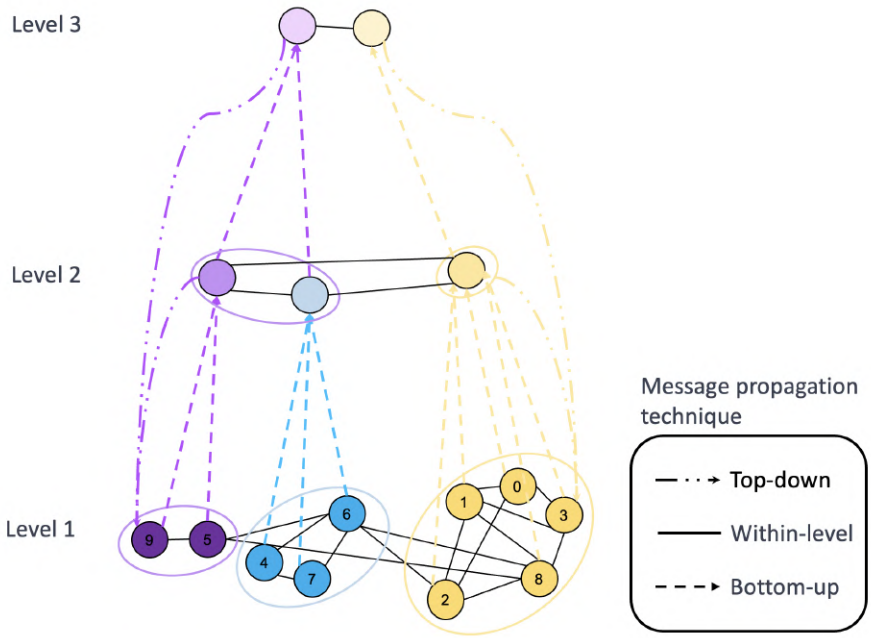


FIGURE 2.5: GCN model architecture

The next stage in the HGNN architecture is the GCN for updating node representations. This comes after the input graph generation and hierarchical structure generation steps. This stage involves training the GCN model using a predetermined number of epochs e,

which determines the number of times the model iteratively updates the node representation based on its neighbors. Setting *e* too low may result in underfitting (the model does not capture an underlying pattern in the data). In contrast, setting *e* too high may lead to overfitting (the model becomes too sensitive, failing to generalize well to unseen data).

The GCN model updates the node representations by aggregating information from their neighborhood through node aggregation and message passing operations. This is done iteratively with the updated node representations used in the subsequent iterations to further refine the representations and capture higher-level features and patterns in the input graph.

2.2.1.4 *Evaluation and updating (i number of iteration)*

The updating step is an iterative process where the inferred graph is passed back into HGNN. This iterative process starts from the hierarchical structure generation to updating and evaluation stage until convergence is reached. This is done to refine the HGNN model and achieve the desired performance. Overall, the updating step plays an important role in ensuring that the HGNN model is effective in capturing the underlying network structure and achieving the desired task objective. After the set change in total loss criterion for convergence or maximum number of iteration is met, the effectiveness of HGNN is be assessed through some performance evaluation metrics.

2.2.2 *Loss Function*

To tackle the challenge of learning complex Gene Regulatory Networks using Graph Neural Networks, it is important to have a suitable loss function that can enhance the overall performance of the GNN algorithm. Therefore, our HGNN algorithm uses a loss function composed of four components, each designed to contribute to the optimization process.

2.2.2.1 *Binary Cross Entropy(BCE)*

Mathematically, this is defined by,

$$\mathscr{L}(y,\hat{y}) = -\frac{1}{N}\sum_{i=0}^{N}\left[y^{(i)}\log\left(\sigma\left(\hat{y}^{(i)}\right)\right) + \left(1-y^{(i)}\right)\cdot\log\left(1-\sigma\left(\hat{y}^{(i)}\right)\right)\right]$$

where $\hat{y}^{(i)}$ represent predicted edge weights. These weights are also referred to as logits. $\sigma(\cdot)$ is a sigmoid function $\left(\sigma(\hat{y}) = \frac{1}{1+\exp^{-\hat{y}}} \in [0,1]\right)$ and $y^{(i)}$ represent the true edge weight. We utilize this component of the loss to control edge weight predictions.

2.2.2.2 *Spectral Mean Squared Error (SMSE)*

The loss function's spectral mean squared error component is defined based on the spectrum $\gamma\left(\hat{G}\right)$ and $\gamma\left(G\right)$ of graphs $\hat{G}$ and G respectively (Wilson and Zhu, 2008; Wills and Meyer, 2020). Mathematically,

$$SMSE = \frac{1}{N}\sum_{i=1}^{N}\left(\lambda_i - \hat{\lambda}_i\right)^2$$

where λ_i and $\hat{\lambda}_i$ represent eigenvalues of the matrix representation of G and $\hat{G}$ respectively. Here, the matrix representation used is the signless laplacian matrix chapter (2.1.3). The purpose of using this component is to achieve stabilization for the entire hierarchy, including both the bottom and super layer nodes.

2.2.2.3 *Clustering Loss (CL)*

This loss component seeks to improve the clustering performance of HGNN. Mathematically,

$$\begin{aligned} CL &= MS_{\text{within}} - MS_{\text{between}} \\ &= \frac{1}{n-k} \cdot \sum_{i=1}^{k} \sum_{j=1}^{|c_i|} \left(x_{ij} - \bar{x}_{i.} \right)^2 - \frac{1}{k-1} \cdot \sum_{i=1}^{k} \left(x_{i.} - \bar{x}_{..} \right)^2 \end{aligned}$$

where n is the number of observations, k, the number of inferred clusters, and $|c_i|$ represents the number of observations in a given cluster i. It can be easily seen that $k \leq n$. x_{ij} represents the input data with $\bar{x}_{i.}$ being mean of cluster i, and $\bar{x}_{..}$, the overall mean.

2.2.2.4 *Penalty on the number of clusters(k)*

The purpose of incorporating the penalty on the number of clusters (k) component into the loss function is to stabilize the number of clusters that are inferred.
To get the total loss, these four components are weighted and added. Thus, the total loss, l is given by

$$l = \lambda_1 BCE + \lambda_2 SMSE + \lambda_3 CL + \lambda_4 k$$

where $\lambda_1, \ldots, \lambda_4$ represent weights on each component. In the context of our simulation study and the results presented in the next chapter, we aim to evaluate the impact of each component of the loss function. To this end, we assign a weight of 1 to a single component while setting the weights of the remaining components to 0.

CHAPTER 3

SIMULATION STUDY

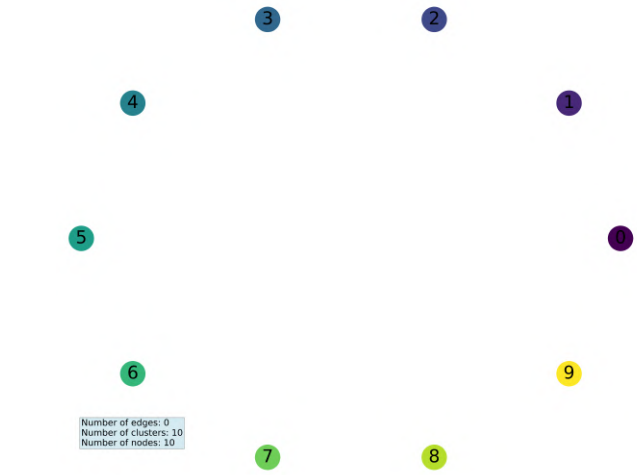
In this chapter, we focus on our simulation study. We will discuss our data set generation procedure. The second part of this chapter presents our results and findings from running HGNN on two main simulated pseudo-gene expression data sets generated from a disconnected two-layer hierarchy and a connected three-layer hierarchy.

3.1 SIMULATION OF HIERARCHICAL GRAPH LAYERS

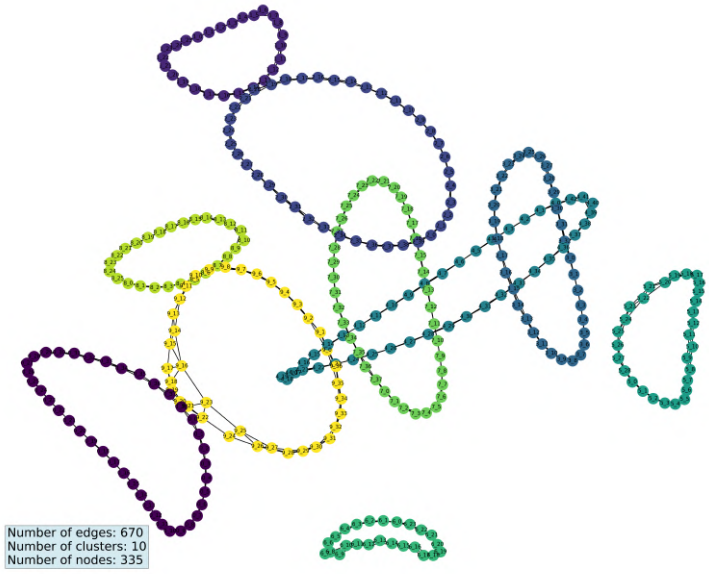
To test our HGNN algorithm, we simulate a hierarchy of layers from which gene expression data is generated and passed to the algorithm. In this section, we go through an overview of the simulation of these hierarchies.

3.1.1 *Disconnected two-layer hierarchy*

In this simulation, the goal was to generate a simple pseudo-gene expression data set. The top layer of this hierarchy consists of ten(randomly chosen) disconnected nodes (also referred to as super nodes). From each of these nodes, we randomly generate a directed small-world network to form the bottom layer of the hierarchy. This results in the bottom layer having ten connected subgraphs. Thus, the final hierarchy comprises ten super nodes in the top layer and ten connected subgraphs in the bottom layer. From the bottom layer, a pseudo-gene expression data set is randomly generated from normal distribution with a specified noise based on the connections existing between nodes .



(A)



(B)

FIGURE 3.1: A simulated hierarchy of two layers with 10 disconnected super nodes and 10 connected subgraphs in the bottom layer. (A) Top layer with 10 disconnected nodes. (B) Bottom layer with 10 connected small-world subgraphs and 670 total number of edges. Node colors are based on the top layer nodes.

3.1.2 *Connected three-layer hierarchy*

The goal of this simulation was to generate a more complex pseudo-gene expression data set to be used as input for the algorithm. Three layers make up the hierarchy of this simulation: the top layer, the middle layer, and the bottom layer. For the top layer, we generated a directed small-world network with 10 nodes and 20 edges. Furthermore, for each node in the top layer, a directed small-world subnetwork is randomly generated. Next, we used these subgraphs to construct the middle layer module by adding edges between the generated subgraphs based on the distribution of edges in the top layer. To add an edge between two nodes, we generated a random number uniformly between 0 and 1 and added the edge if the number was less than a specified connection probability. We repeat the same process to generate the bottom layer module from the middle layer.

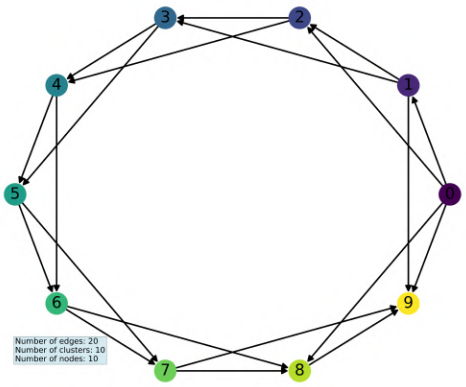


FIGURE 3.2: A simulated hierarchy of three layers - Top layer with 10 nodes and 20 edges

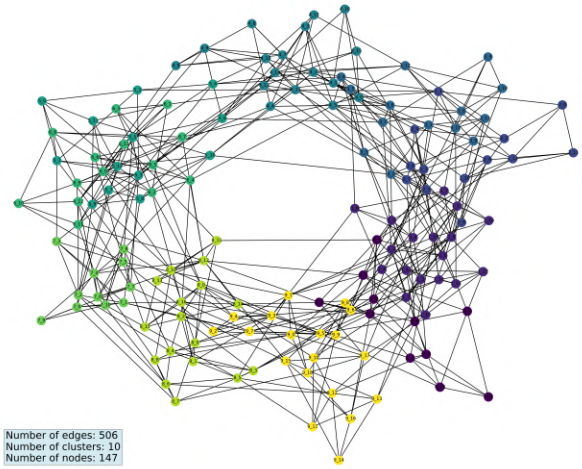


FIGURE 3.3: Continued from Figure 3.2 - Middle layer with 147 nodes and 506 edges. Node colors are based on the top layer nodes.

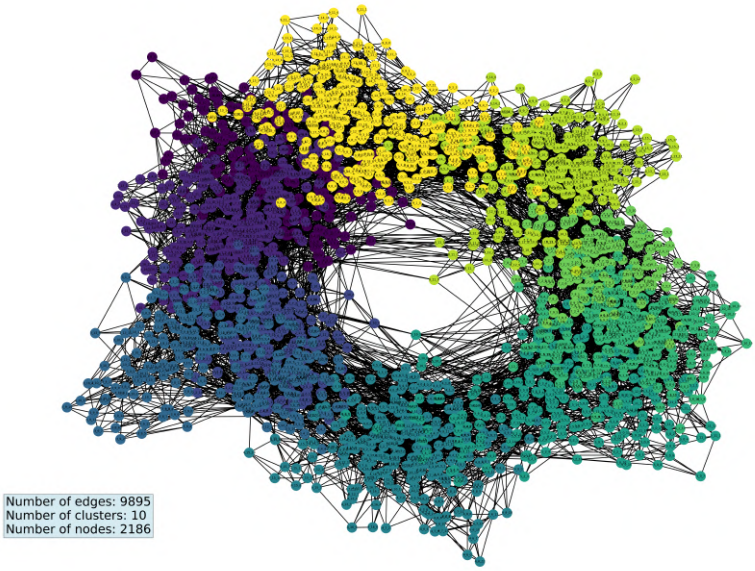


FIGURE 3.4: Continued from Figure 3.3. Bottom layer with 2186 nodes and 9895 edges. Node colors are based on the top layer nodes.

3.1.3 *Gene expression dataset simulation*

After the hierarchies are constructed, this is the last stage of the simulation is to generate the sample gene expression dataset. The bottom layer is used to randomly generate a pseudo-gene expression dataset based on node and edge distribution in the hierarchy. Here, the adjacency matrix and the topological order of the bottom layer are utilized to represent the entire hierarchy. To generate the expression values for each node (representing genes) in the hierarchy, a random normal distribution is also utilized with an initial mean and standard deviation. The mean and standard deviation are crucial in controlling the amount of noise added to the generated gene expression dataset. The generated data is of dimension $m \times n$, where m represents the samples or individuals and n, the genes or nodes. We generate two pseudo-gene expression datasets for each simulated hierarchy, with standard deviations of 0.1 and 0.5, and an initial mean equal to 0.

As an example, suppose A, B, and C of Figure 2.2 represent three different bottom layers. Using the topological order,

- For (A), we obtain the generated pseudo-gene expression for each individual or sample as follows. First, we assign the expression value of node 1 as

$$x_{i1} \sim N\left(\mu = 0, \sigma^2\right),$$

  where x_{i1} denotes the expression value for node 1 for the i^{th} individual, μ is the initial mean (which is set to 0), and σ is the standard deviation specified.

  Since there exists a directed edge from node 1 to node 2, we can use the value of node 1 to calculate the value of node 2. Specifically, we set

$$x_{i2} \sim N\left(\mu = x_{i1}, \sigma^2\right),$$

  where x_{i2} is the expression value for node 2 for the i^{th} individual, μ is the mean given by the value of x_{i1}, and σ is the same standard deviation as before.

Similarly, we obtain the value of node 3 using the value of node 2 as the mean,

$$x_{i3} \sim N\left(\mu = x_{i2}, \sigma^2\right),$$

where x_{i3} is the expression value for node 3 for the i^{th} individual.

- To generate pseudo-gene expression for an individual or sample in (B), we use normal distribution with a specified standard deviation σ and initial mean $\mu = 0$ for nodes 1 and 2, denoted by x_{i1} and x_{i2} respectively. For node 3, we take the average of the expression values for nodes 1 and 2, denoted by $\bar{x}_{i\{12\}}$, and set it as the mean for the normal distribution, resulting in the expression value

$$x_{i3} \sim N\left(\mu = \bar{x}_{i\{12\}}, \sigma^2\right).$$

- The generated pseudo-gene expression for an individual or sample for (C) will also be given by

$$x_{i1} \sim N\left(\mu = 0, \sigma^2\right),$$

where x_{i1} is the expression value for node 1 for the i^{th} individual, μ is the initial mean and σ is the specified standard deviation.

Furthermore, the expression values for nodes 2 and 3 are given by

$$x_{i2} \sim N\left(\mu = x_{i1}, \sigma^2\right)$$
$$x_{i3} \sim N\left(\mu = x_{i1}, \sigma^2\right),$$

where x_{i2} and x_{i3} are the expression values for nodes 2 and 3 for the i^{th} individual, μ is the mean given by x_{i1}, and σ is the specified standard deviation.

In summary, the expression values of x_{i1}, x_{i2}, and x_{i3} are generated sequentially based on the topological order, with each node's value being drawn from a normal distribution with mean equal to the value of its parent node and the same specified standard deviation σ for all nodes.

3.2 ANALYSIS

3.2.1 *Quantitative Evaluation*

In this section, we briefly discuss examining the different components of our loss function. We adopt three main quantitative evaluation metrics from Rosenberg and Hirschberg (2007) in our analysis to measure the performance of our algorithm, HGNN. These include;

3.2.1.1 *Homogeneity ($\mathscr{H}$)*

We use this metric to measure how similar nodes (genes) are in a cluster. In other words, the distribution of nodes within a cluster should be biased towards a particular class of nodes. It is mathematically defined as,

$$\begin{aligned}\mathscr{H} &= 1 - \frac{H(C|K)}{H(C)} \\ &= 1 - \left[\frac{-\sum\limits_{c\in C, k\in K} \frac{n_{ck}}{N} \log\left(\frac{n_{ck}}{n_k}\right)}{-\sum\limits_{i} p(c_i)\log\big(p(c_i)\big)} \right]\end{aligned}$$

where $H(C)$ is the entropy (Shannon and Weaver, 1949) related to cluster C, n_{ck} represents the number of nodes (genes) of class c in cluster k, n_k is the number of nodes (genes) in cluster k. It can be easily seen that when all nodes (genes) in class c are in cluster k, $n_{ck} = n_k$.

Thus,

$$\begin{aligned}
\mathscr{H} &= 1 - \left[\frac{-\sum\limits_{c\in C, k\in K} \frac{n_{ck}}{N} \log\left(\frac{n_k}{n_k}\right)}{-\sum\limits_i p(c_i)\log\big(p(c_i)\big)} \right] \\
&= 1 - \left[\frac{-\sum\limits_{c\in C, k\in K} \frac{n_{ck}}{N} \log(1)}{-\sum\limits_i p(c_i)\log\big(p(c_i)\big)} \right] \\
&= 1 - \left[\frac{0}{-\sum\limits_i p(c_i)\log\big(p(c_i)\big)} \right] \\
&= 1.
\end{aligned}$$

$\mathscr{H} \in [0,1]$ with 1 representing a perfect score.

3.2.1.2 *Completeness ($\mathscr{C}$)*

This metric measures the degree to which similar nodes are clustered together in the same cluster. Mathematically,

$$\begin{aligned}
\mathscr{C} &= 1 - \frac{H(K|C)}{H(K)} \\
&= 1 - \left[\frac{-\sum\limits_{c\in C, k\in K} \frac{n_{ck}}{N} \log\left(\frac{n_{ck}}{n_c}\right)}{-\sum\limits_i p(k_i)\log\big(p(k_i)\big)} \right]
\end{aligned}$$

where $H(K)$ is the entropy (Shannon and Weaver, 1949) associated with cluster K, n_c is the number of nodes (genes) in with label (belonging to) c. We can easily deduce that $\mathscr{C} = 1$

when all nodes (genes) in cluster k belongs to class c (that is, $n_{ck} = n_c$).

$$\begin{aligned}
\mathscr{C} &= 1 - \left[\frac{-\sum\limits_{c\in C, k\in K} \frac{n_{ck}}{N} \log\left(\frac{n_c}{n_c}\right)}{-\sum\limits_i p(k_i)\log\big(p(k_i)\big)} \right] \\
&= 1 - \left[\frac{-\sum\limits_{c\in C, k\in K} \frac{n_{ck}}{N} \log(1)}{-\sum\limits_i p(k_i)\log\big(p(k_i)\big)} \right] \\
&= 1 - \left[\frac{0}{-\sum\limits_i p(k_i)\log\big(p(k_i)\big)} \right] \\
&= 1.
\end{aligned}$$

$\mathscr{C} \in [0,1]$ with 1 representing a perfect score.

3.2.1.3 *Normalized Mutual Information (NMI)*

Mathematically defined as

$$NMI = 2 \times \frac{\mathscr{H} \times \mathscr{C}}{\mathscr{H} + \mathscr{C}}$$

where $\mathscr{H}$ and $\mathscr{C}$ represent homogeneity and completeness, respectively. The NMI also provides a quantitative way to assess the quality of our algorithm. $NMI \in [0,1]$ with 1 representing a perfect clustering quality.

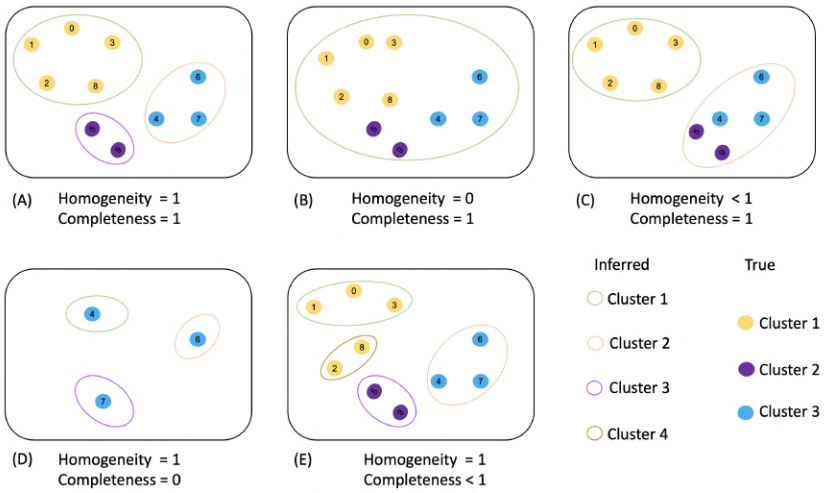


FIGURE 3.5: Illustration of the metrics: completeness and homogeneity. A). Both metrics are perfect. As can be seen, each inferred cluster only has one class from the truth, and all the nodes from the true clusters are inferred to be in the same cluster. B). Here, homogeneity = 0 as all nodes from the true clusters are inferred to be in one cluster. Completeness = 1 (all nodes from the same true cluster are inferred to belong to the same cluster). C). We observe that all nodes belonging to a true cluster are placed in the same inferred cluster, thus completeness = 1. Since an inferred cluster is made of nodes from two true clusters (2 and 3), homogeneity < 1. D). Each inferred cluster contains only nodes from a true cluster, thus, homogeneity = 1 but completeness = 0 as nodes from a true cluster belong to different inferred clusters. E). Completeness < 1 (we observe that nodes from true cluster 1 are inferred to belong to different clusters) with homogeneity =1 (each inferred cluster has one class of nodes from the true clusters).

3.2.1.4 *Evaluation of hierarchy*

For evaluating the hierarchy, we use the number of super layers which represent the number of upper levels in the hierarchy and super nodes which represent the number of upper level nodes in the hierarchy. We make a comparison between the true and inferred hierarchy from HGNN. Additionally, we use the number of true and inferred clusters as a means of comparison to check the performance of HGNN. These metrics also allow us to compare the true and inferred hierarchy from HGNN and determine the degree of similarity between the two.

3.2.2 *Visualization*

Visualizations play an important role in analyzing and interpreting results. It helps identify possible patterns and structures. Here, we discuss four main visualizations used to analyze the result from running HGNN.

- Loss function plot over iterations. The loss function plot over iterations shows the nature of the loss curve over iterations. This visualization helps evaluate the performance of HGNN and identify possible convergence issues or optimal stopping points.

- Inferred cluster network with true cluster labels. This visualization involves comparing the true cluster labels with the inferred cluster labels. To do this, the inferred cluster network is visualized with node coloring using the true cluster labels. This provides a better idea of the performance of HGNN and helps identify any misclassifications or overlap between clusters. We expect each cluster in the inferred to have the same node colors as the algorithm produces a perfect inference.

- Adjacency heatmap of true and inferred. The adjacency heatmap is a binary matrix visualization of the true network and inferred network/clusters at the gene level. To provide better visualization, the rows and columns are sorted based on the true cluster labels. This helps identify clusters that are not apparent from the inferred cluster network.

- Adjacency heatmap of true and inferred cluster evolution. The heatmap of true and inferred cluster evolution shows the similarity between true and inferred cluster labels over iterations. This helps to visualize inferred cluster label distribution over iteration in comparison to the true labels for each component of the loss function.

3.3 RESULTS

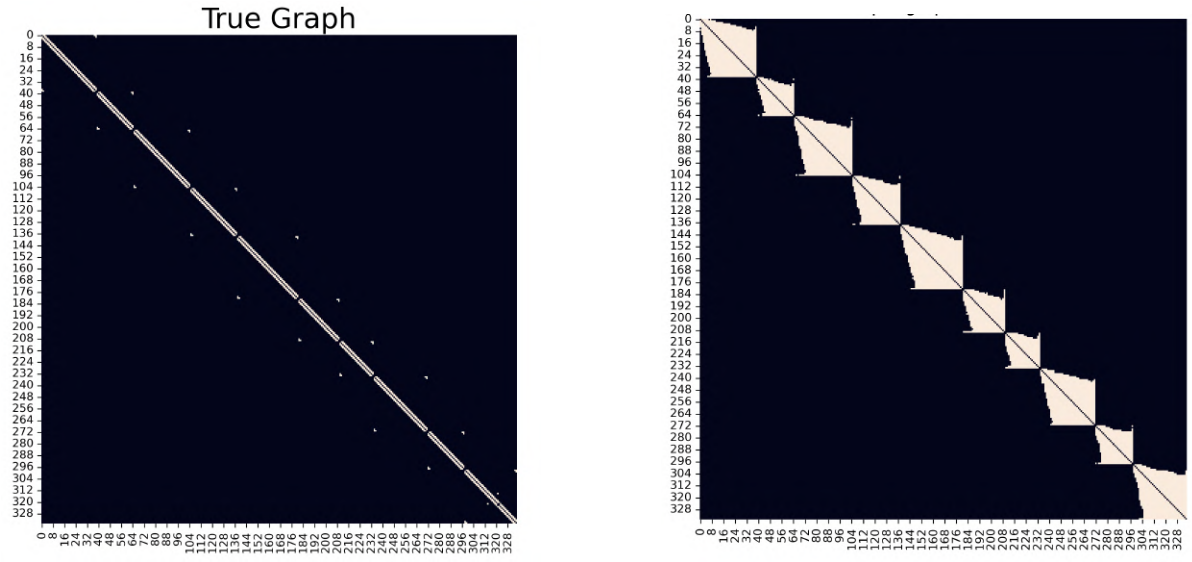
This section presents the results of our simulations. We show the results of our simulations after running HGNN on our simulated pseudo-gene expression data sets with standard deviations 0.1 and 0.5 for our disconnected and connected hierarchy.

3.3.1 *Disconnected two-layer hierarchy*

We show the result of HGNN on our simple pseudo- gene expression dataset with a standard deviation of 0.1. Here in our implementation, we set the change in total loss criterion to 0.01 and limit the maximum number of iterations to 10, with the number of within epoch loops set to 50. The absolute correlation threshold was also set to 0.5. The value of the threshold was randomly chosen for this specific simulation.

3.3.1.1 *Disconnected small-world two-layer hierarchy (standard deviation = 0.1)*

This section shows the results of HGNN on a simple pseudo-gene expression dataset with a standard deviation of 0.5.



(A) (B)

FIGURE 3.6: Binary heatmap of simulated network adjacency(A) and correlation inferred adjacency(B) matrix. Nodes were sorted based on the true labels.

The binary heatmap of the adjacency matrix for the true bottom layer and the absolute correlation adjacency are displayed in Figure 3.6. It is evident that both heatmaps exhibit ten distinct clusters, indicating that the inferred adjacency matrix captures a significant portion of the underlying network structure. However, there are numerous edges inferred in the correlation adjacency identified by broader clusters in the heatmap(Figure 3.6b). Overall, these results suggest that the inferred adjacency matrix is a good representation of the true bottom-layer network.

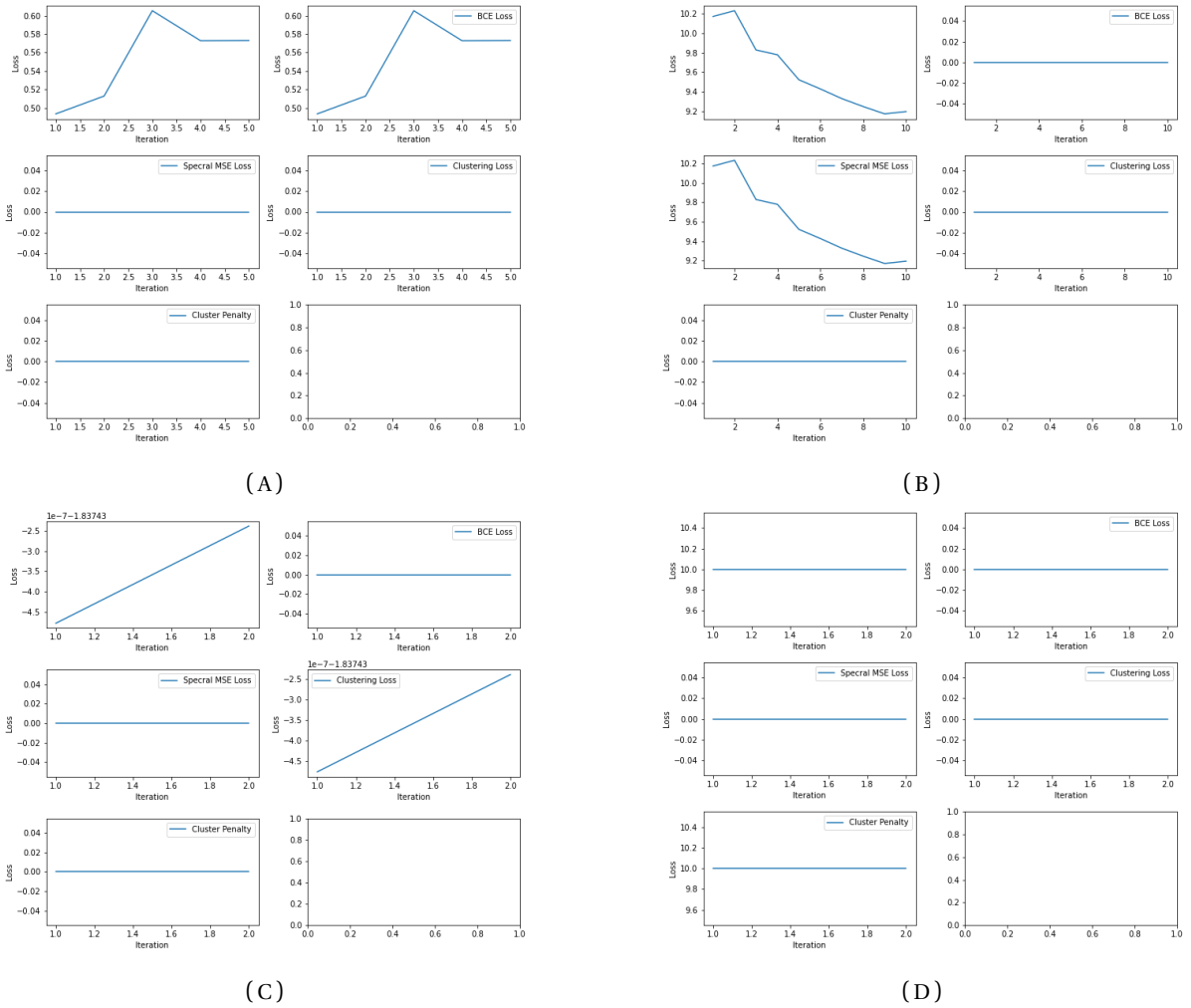


FIGURE 3.7: Hierarchy of 2 disconnected layers: Visualization of individual component of the loss function. (A) Binary Cross Entropy Loss curve. (B) Spectral Mean Square Error loss curve. (C) Clustering loss curve. (D) Penalty on the clustering curve.

Figure 3.7 provides a visualization of the individual loss components in the context of our proposed HGNN. Specifically, the BCE loss is shown in Figure 3.7a, the SMSE loss in Figure 3.7b, and the clustering loss in Figure 3.7c. It is observed that the BCE loss curve shows an initial rise, followed by a gradual decrease, but the first iteration remains the minimum loss value. In contrast, the SMSE loss curve decreases steadily and reaches the minimum

loss value at the ninth iteration. Lastly, the clustering loss curve shows a consistent rise throughout the iterations, with the first iteration having the minimum loss value.

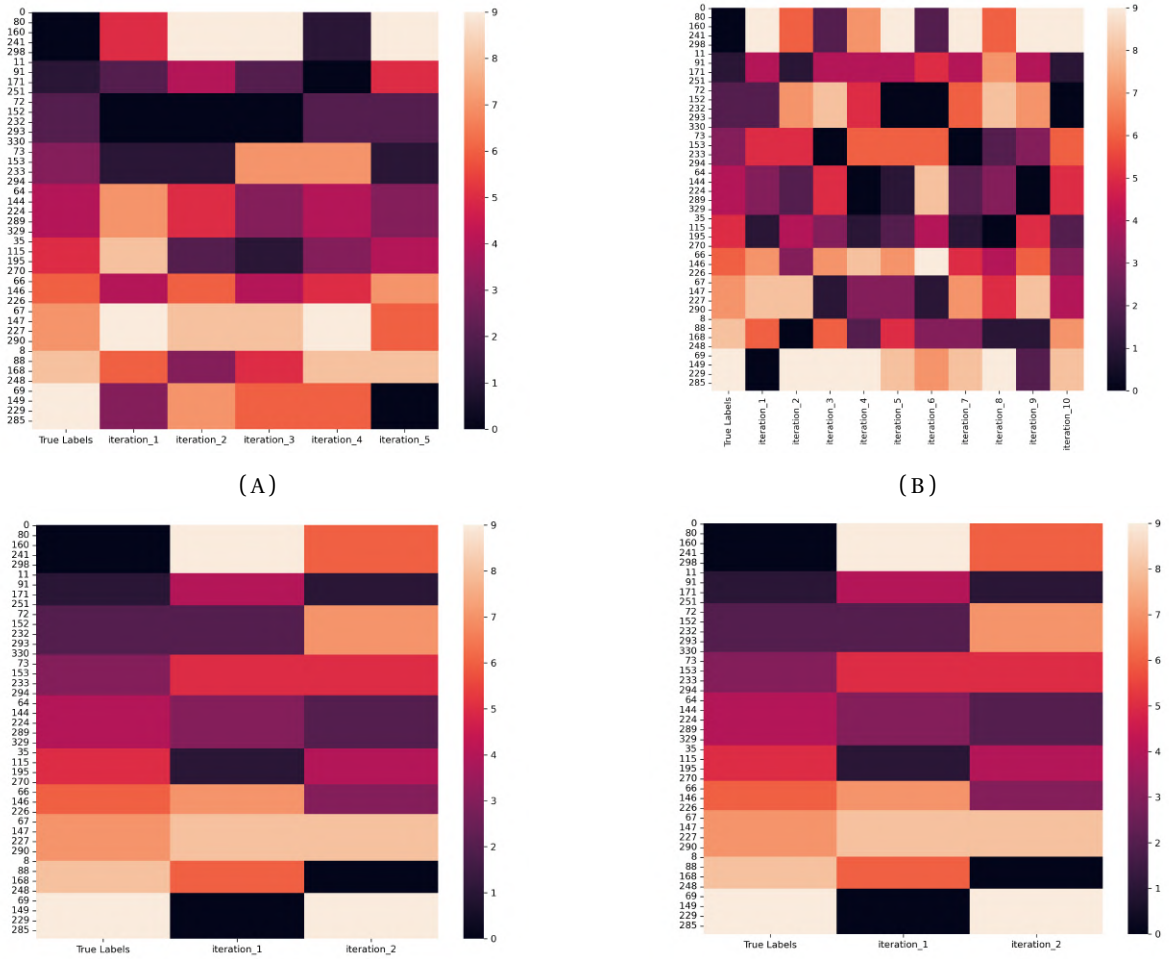


FIGURE 3.8: Visualization of inferred cluster labels and true cluster labels over iterations. (A) Cluster evolution using BCE loss. (B) Cluster evolution using SMSE loss. (C) Cluster evolution using Clustering loss. (D) Cluster evolution using cluster penalty loss.

Figure 3.8 shows the true labels with the inferred labels at each iteration for each loss component. Notably, all the individual components of the loss exhibit good cluster partitioning, as identified by the clear separation of the inferred labels in the plot. The inferred labels align well with the true labels from the top layer, indicating the effectiveness and

robustness of HGNN. Overall, the good cluster partitioning observed in the individual loss components reinforces the validity and accuracy of the HGNN in identifying the underlying network structure in the simple pseudo-gene expression dataset.

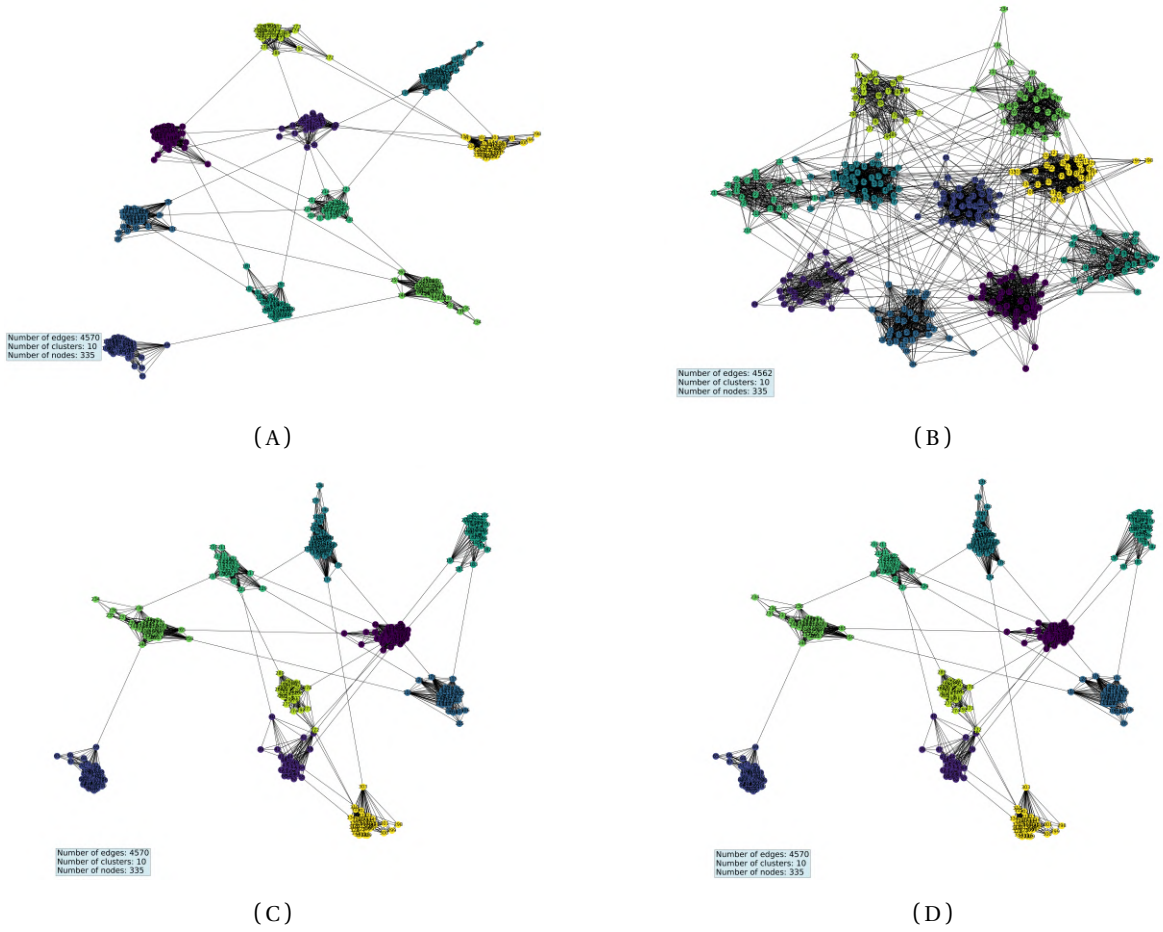


FIGURE 3.9: Visualization of the inferred cluster at convergence of HGNN. (A)Inferred cluster using BCE loss. (B) Inferred cluster using SMSE loss. (C) Inferred cluster using clustering loss. (D) Inferred cluster using cluster penalty loss

Figure 3.9 gives a visualization of the inferred bottom layer graph structure under each loss component. The node coloring is based on true cluster labels. We observed that for all loss components, the true cluster labels have been accurately placed in the same class, and all classes are of the same kind. This visualization of the inferred bottom layer graph

structures gives insight into the performance of HGNN and its ability to accurately capture the underlying network structure.

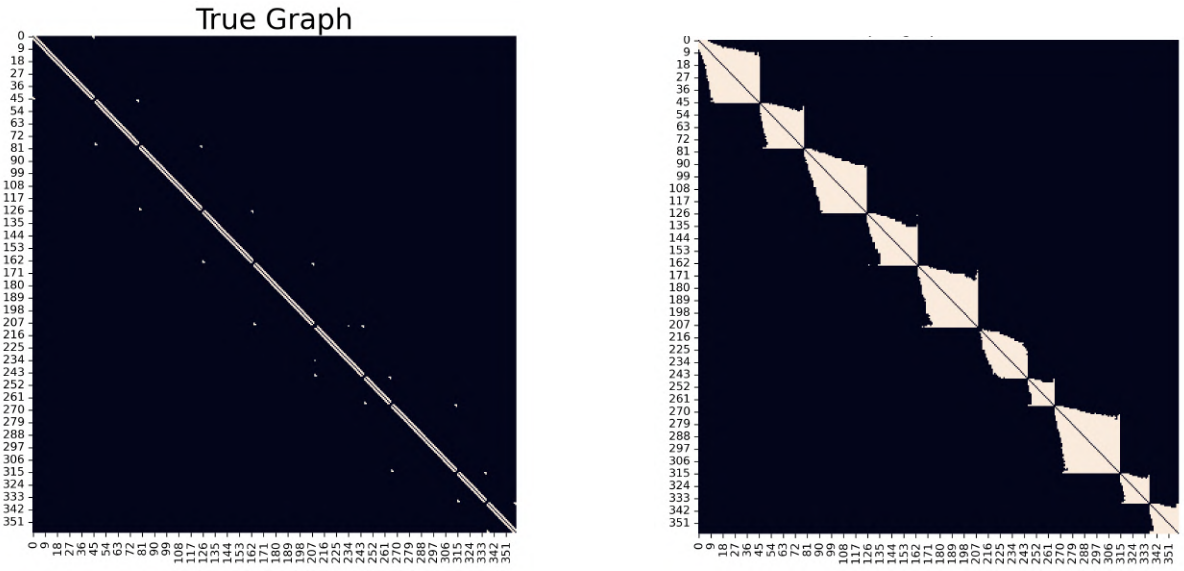
Loss Function Parameters				Clustering Performance			Inferred Cluster Information			
BCE	SMSE	CL	K	Homogeneity	Completeness	NMI	Clusters	Super nodes	Super Edges	Super Layer
0.4934	0	0	0	1	1	1	10	10	0	1
0	9.1716	0	0	1	1	1	10	10	43	1
0	0	-1.8374	0	1	1	1	10	10	0	1
0	0	0	10	1	1	1	10	10	0	1

TABLE 3.1: Performance metric of HGNN on our simple pseudo-gene expression dataset generated from a disconnected small-world two-layers with standard deviation 0.1.

Table 3.1 shows the clustering performance metrics of HGNN on the pseudo-gene expression dataset generated from a disconnected small-world 2-layer hierarchy with a standard deviation of 0.1. The loss function parameters include Binary Cross-Entropy (BCE), Square Mean Square Error (SMSE), Clustering Loss (CL), and penalty on the number of clusters (k). The clustering performance metrics are homogeneity, completeness, and Normalized Mutual Information (NMI). The inferred cluster information comprises the number of clusters, super nodes, super edges, and super layers. HGNN performs well in terms of clustering performance with a perfect score (1) for all components of the loss. Similarly, the inferred cluster information produces an accurate or perfect result in terms of inferring the number of clusters (10), the number of super nodes (10), the number of super edges (0), and the number of super-layers (1). We observe that for the SMSE loss, the number of inferred edges between top layer nodes were 43. These results suggest that HGNN can accurately infer underlying graph structure in the data.

3.3.1.2 *Disconnected small-world two-layer hierarchy (standard deviation = 0.5)*

This section shows the results of HGNN on a simple pseudo-gene expression dataset with a standard deviation of 0.5.



(A) (B)

FIGURE 3.10: Binary heatmap of simulated network adjacency(A) and correlation inferred adjacency(B) matrix. Nodes were sorted based on the true labels.

Figure 3.10 displays the binary heatmaps of the adjacency matrices for the true bottom layer and the inferred absolute correlation adjacency in HGNN. Ten distinct clusters are visible in both heatmaps, indicating that the inferred adjacency matrix captures a significant portion of the underlying network structure. Though more edges seem to be inferred by the correlation adjacency, we expect the GCN step of HGNN to down-weight these false edges.

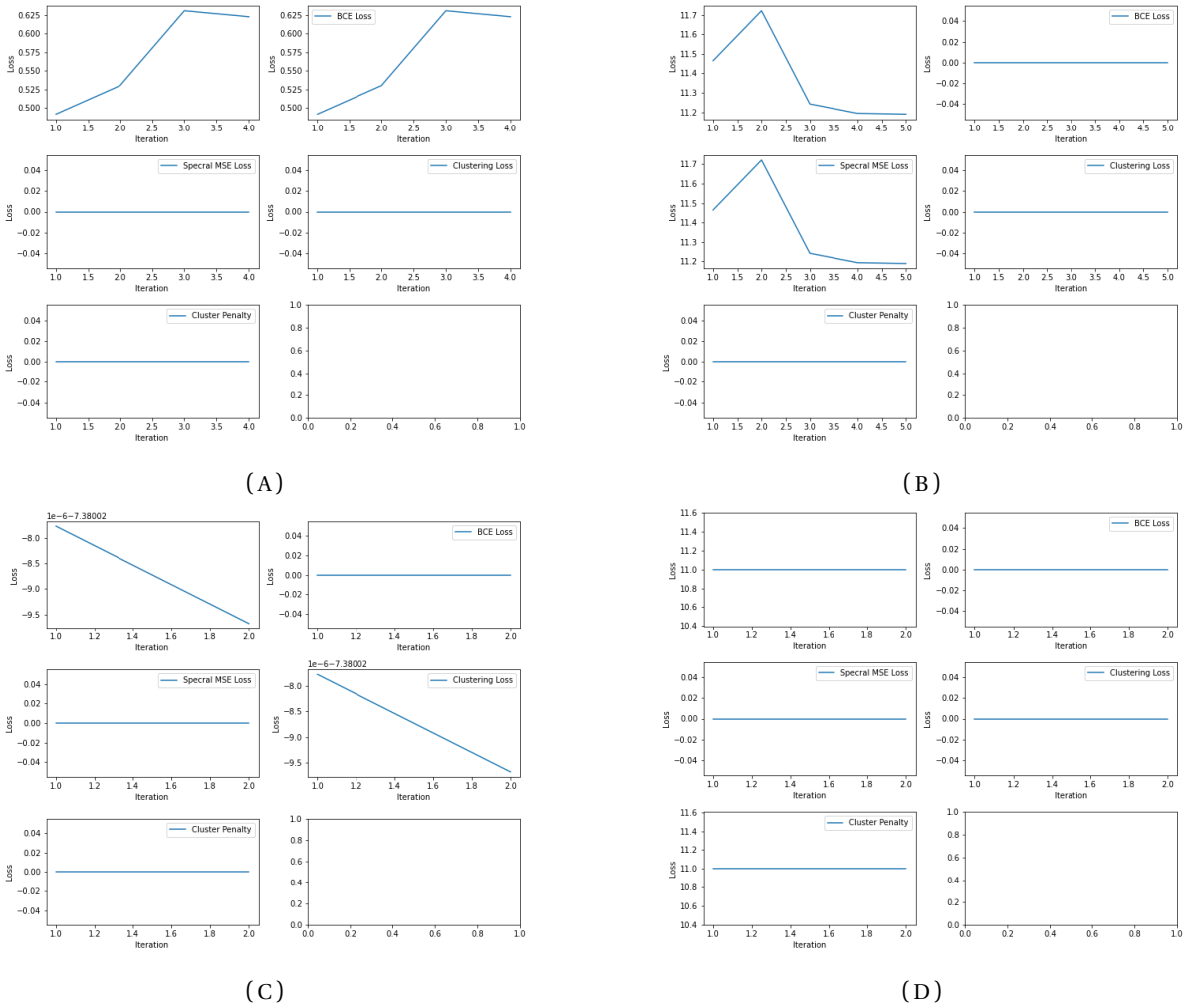
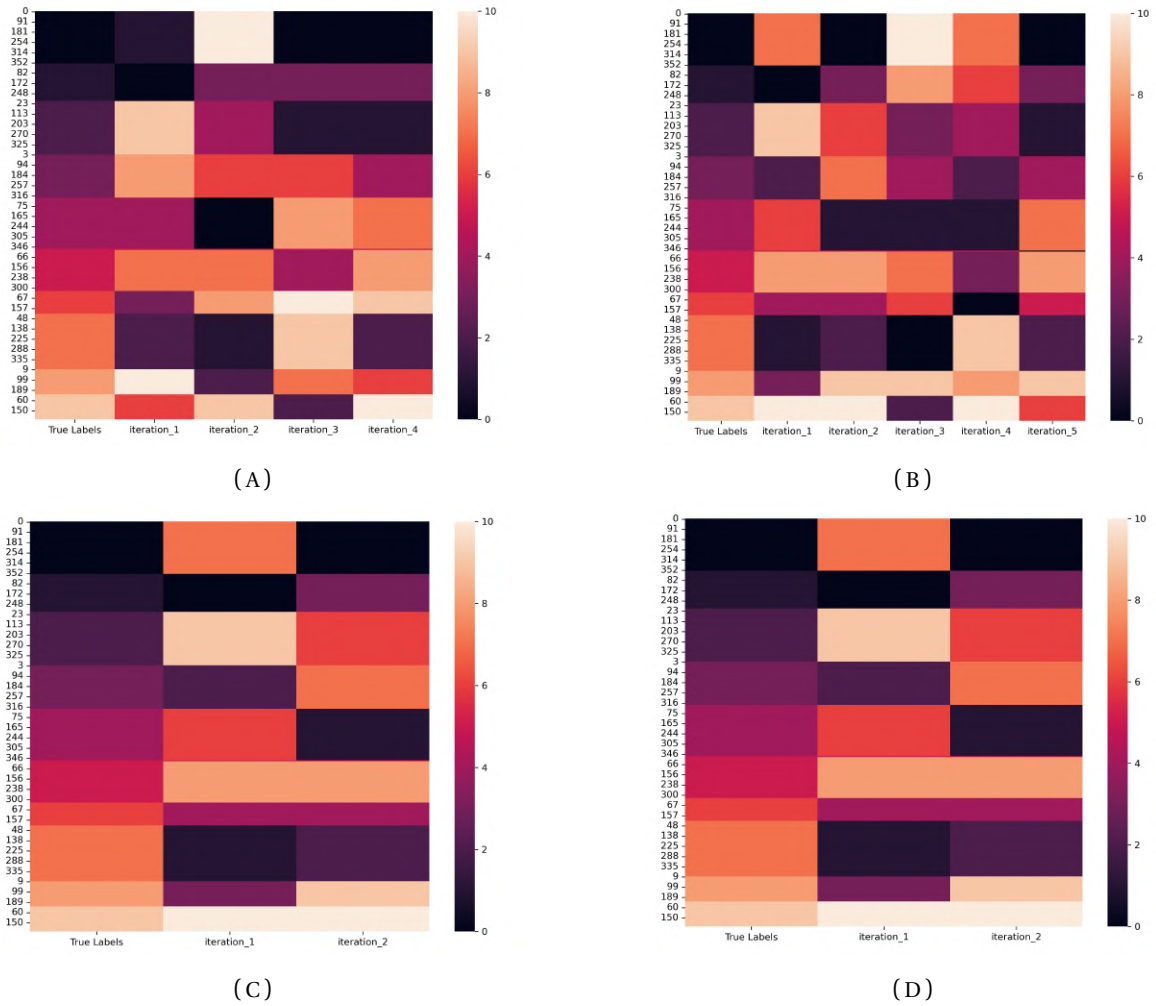


FIGURE 3.11: Hierarchy of 2 disconnected layers: Visualization of individual component of the loss function. (A) Binary Cross Entropy Loss curve, (B) Spectral Mean Square Error loss curve. (C) Clustering loss curve. (D) Penalty on the clustering curve.

Figure 3.11 gives a visualization of the individual loss components of HGNN. Analysis of the loss curves shows that the BCE loss curve initially rises sharply, and then tries to decrease, with the minimum loss value at the first iteration. On the other hand, the SMSE loss curve has a decreasing curve (except from iterations 1 to 2) with the minimum loss value at the fifth iteration. The clustering loss curve also shows a decrease throughout the iterations, with the minimum loss value occurring at the first iteration.



(A) (B)

(C) (D)

FIGURE 3.12: Visualization of inferred cluster labels and true cluster labels over iterations. (A) Cluster evolution using BCE loss. (B) Cluster evolution using SMSE loss. (C) Cluster evolution using Clustering loss. (D) Cluster evolution using cluster penalty loss.

In Figure 3.12, the inferred labels for each loss component are plotted against the true labels at each iteration. The plot clearly shows good cluster partitioning for all individual loss components, with the inferred labels aligning well with the true cluster labels. This result is similar to the result shown in Figure 3.8. Overall, this highlights the capability of HGNN to identify network structures in the disconnected two-layer gene expression data set.

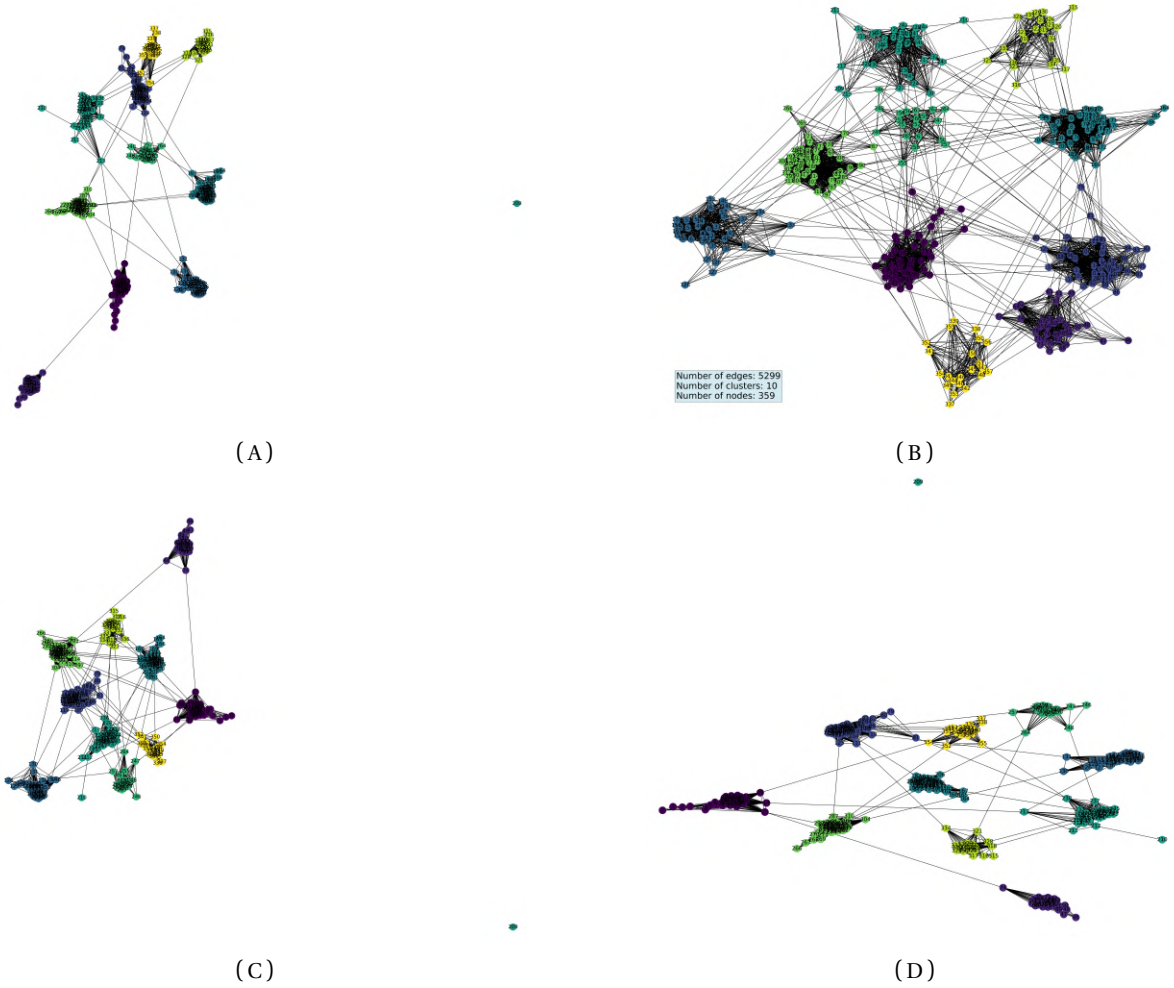


FIGURE 3.13: Visualization of the inferred cluster at convergence of HGNN. (A)Inferred cluster using BCE loss. (B) Inferred cluster using SMSE loss. (C) Inferred cluster using clustering loss. (D) Inferred cluster using cluster penalty loss

Figure 3.13 displays the inferred bottom layer graph structure for each loss component, with the node coloring based on true cluster labels. This visualization shows that the inferred structures accurately capture the true cluster labels, with all true cluster labels placed in the same class and all classes being of the same kind.

Loss Function Parameters				Clustering Performance			Inferred Cluster Information			
BCE	SMSE	CL	K	Homogeneity	Completeness	NMI	Clusters	Super nodes	Super Edges	Super Layer
0.4913	0	0	0	1	0.9944	0.9972	10	10	0	1
0	11.1899	0	0	1	1	1	10	10	44	1
0	0	-7.38	0	1	0.9944	0.9972	10	10	0	1
0	0	0	11	1	0.9944	0.9971	10	10	0	1

TABLE 3.2: Performance metric of HGNN, on our simple pseudo-gene expression dataset using a disconnected small-world two-layers with standard deviation 0.5

Table 3.2 shows the clustering performance metrics of HGNN on the pseudo-gene expression dataset generated from a disconnected small-world two-layer hierarchy with a standard deviation of 0.5. We observe that HGNN performs well in terms of clustering performance(i.e. homogeneity, completeness, NMI) with scores approximately equal to 1 for each component of the loss. Similarly, the inferred cluster information produces an accurate or perfect result in terms of inferring the number of clusters (10), the number of super nodes (10), the number of super edges (0), and the number of super-layers (1). We observe that for the SMSE loss, the number of inferred super edges was 44. These results suggest that HGNN can accurately infer underlying graph structure in the data.

3.3.1.3 *Disconnected scale-free two layer hierarchy (standard deviation = 0.1 and 0.5)*

Similar to the disconnected small-world two-layer hierarchy generated datasets, we test HGNN on a generated dataset with scale-free graph properties. This shows similar patterns in the visualization and performance metric evaluation for the different components of the loss function.

3.3.1.4 *Summary*

Overall, the performance evaluation of HGNN on our disconnected two-layer hierarchy pseudo-gene expression dataset demonstrates its effectiveness in identifying the correct clusters for both small-world and scale-free graph structures with standard deviations of 0.1 and 0.5. Here, the clustering performance metrics were approximately equal to 1, de-

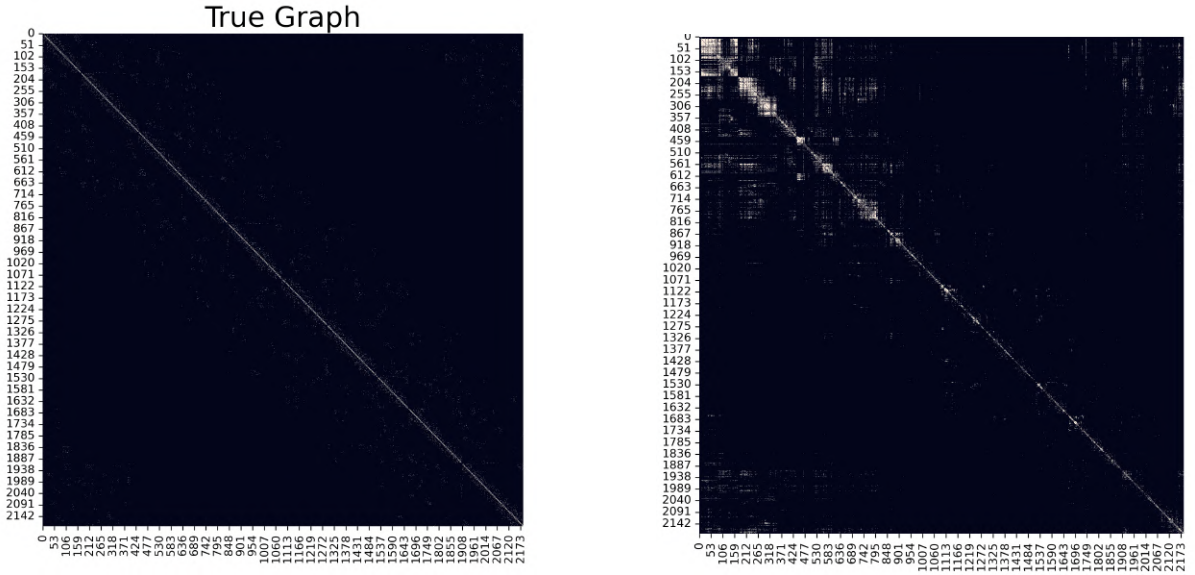
picting a perfect clustering score on datasets. Similarly, the number of inferred super nodes, and super edges for all component of the loss matches the truth except for the SMSE loss inferring false edges in the top layer. It can observed that the individual component of the loss function for the two graph structures produced nearly same results for both clustering performance metrics and the inferred hierarchy information. This highlights the potential of HGNN as a powerful tool for clustering and analyzing gene expression datasets and other network datasets with similar properties.

3.3.2 *Connected three-layer hierarchy*

This section presents the results for running HGNN on our complex pseudo-gene expression dataset generated from a connected three-layer hierarchy with standard deviations of 0.1 and 0.5. Our implementation involves setting the maximum number of iterations to 10 with a change in loss criterion set to 0.01 and performing 50 within-epoch loops. Additionally, we randomly set the absolute correlation threshold to 0.5 to determine the inferred correlation adjacency matrix.

3.3.2.1 *Connected small-world three-layer hierarchy (standard deviation = 0.1)*

In this experiment, we evaluate the performance of HGNN on our complex pseudo-gene expression dataset generated from a connected small-world three-layer hierarchy with a standard deviation of 0.1.



(A) (B)

FIGURE 3.14: Binary heatmap of simulated network adjacency and correlation inferred adjacency matrix. Both true adjacency binary heatmap (A) and correlation adjacency binary heatmap (B) are sorted matrices based on the true cluster the nodes belong to

Figure 3.14a and Figure 3.14b show the binary heatmap of the adjacency matrix for the true bottom layer and the inferred absolute correlation adjacency, respectively, in our complex pseudo-gene expression dataset with standard deviation 0.1. As depicted (in Figure 3.14) , the inferred adjacency matrix captures a some of the underlying network structure in the true adjacency heatmap. However, there are numerous edges inferred in the correlation adjacency. Nevertheless, the overall results suggest that the inferred adjacency matrix is a reasonable representation of the true adjacency.

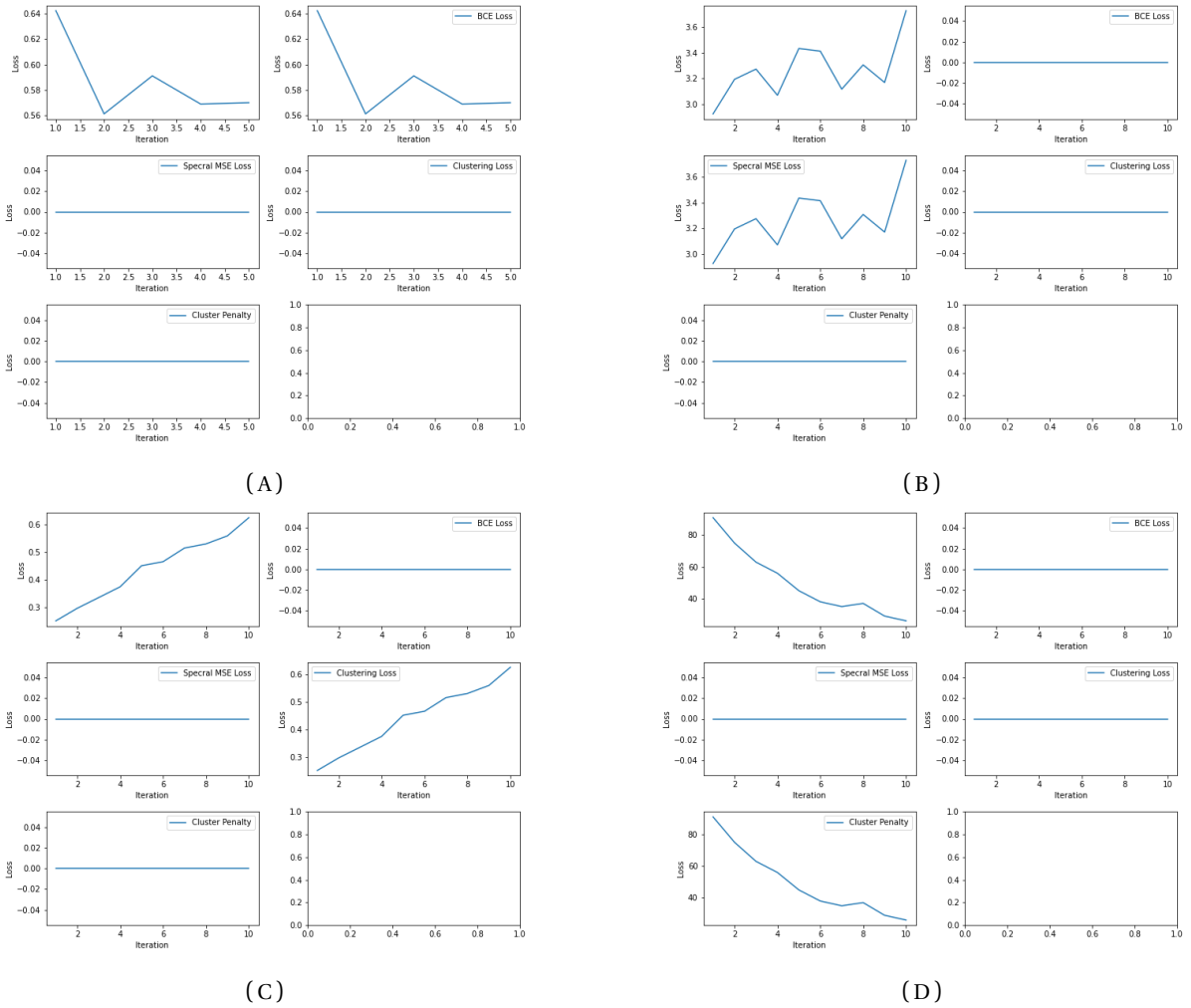


FIGURE 3.15: Hierarchy of 3 connected layers: Visualization of individual component of the loss function. (A) Binary Cross Entropy Loss curve. (B) Spectral Mean Square Error loss curve. (C) Clustering loss curve. (D) Penalty on the clustering curve.

In Figure 3.15, we visualize the individual loss components of HGNN. The BCE loss curve depicted in Figure 3.15a experiences a sharp drop initially, followed by a gradual decrease, but the second iteration still has the minimum loss value. On the other hand, the SMSE (Figure 3.15b) and clustering (Figure 3.15c) loss curves show a steady increase with the minimum loss value at the first iterations, while the penalty on the number of inferred clusters in the Figure 3.15d loss curve decreases over iterations.

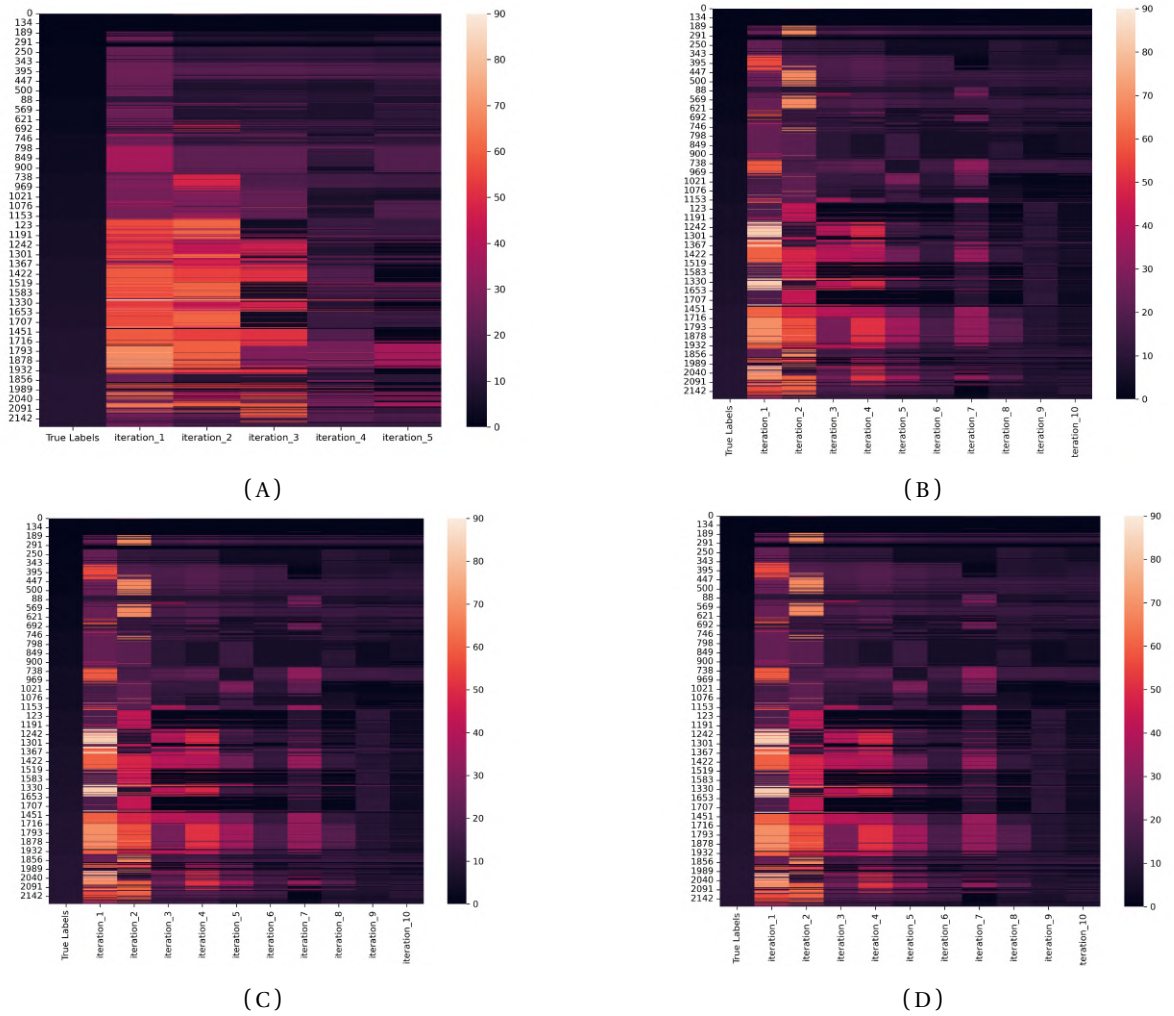


FIGURE 3.16: Visualization of inferred cluster labels and true cluster labels over iterations. (A) Cluster evolution using BCE loss. (B) Cluster evolution using SMSE loss. (C) Cluster evolution using Clustering loss. (D) Cluster evolution using cluster penalty loss.

In Figure 3.16, we see the true labels alongside the inferred labels for each loss component at each iteration. One notable observation is that, for all the individual components of the loss function, true cluster labels from the top layer have been broken into smaller clusters. This leads to having many inferred clusters relative to the truth. Thus here, HGNN has difficulty in making accurate inference with our complex dataset using only individual components of the loss function.

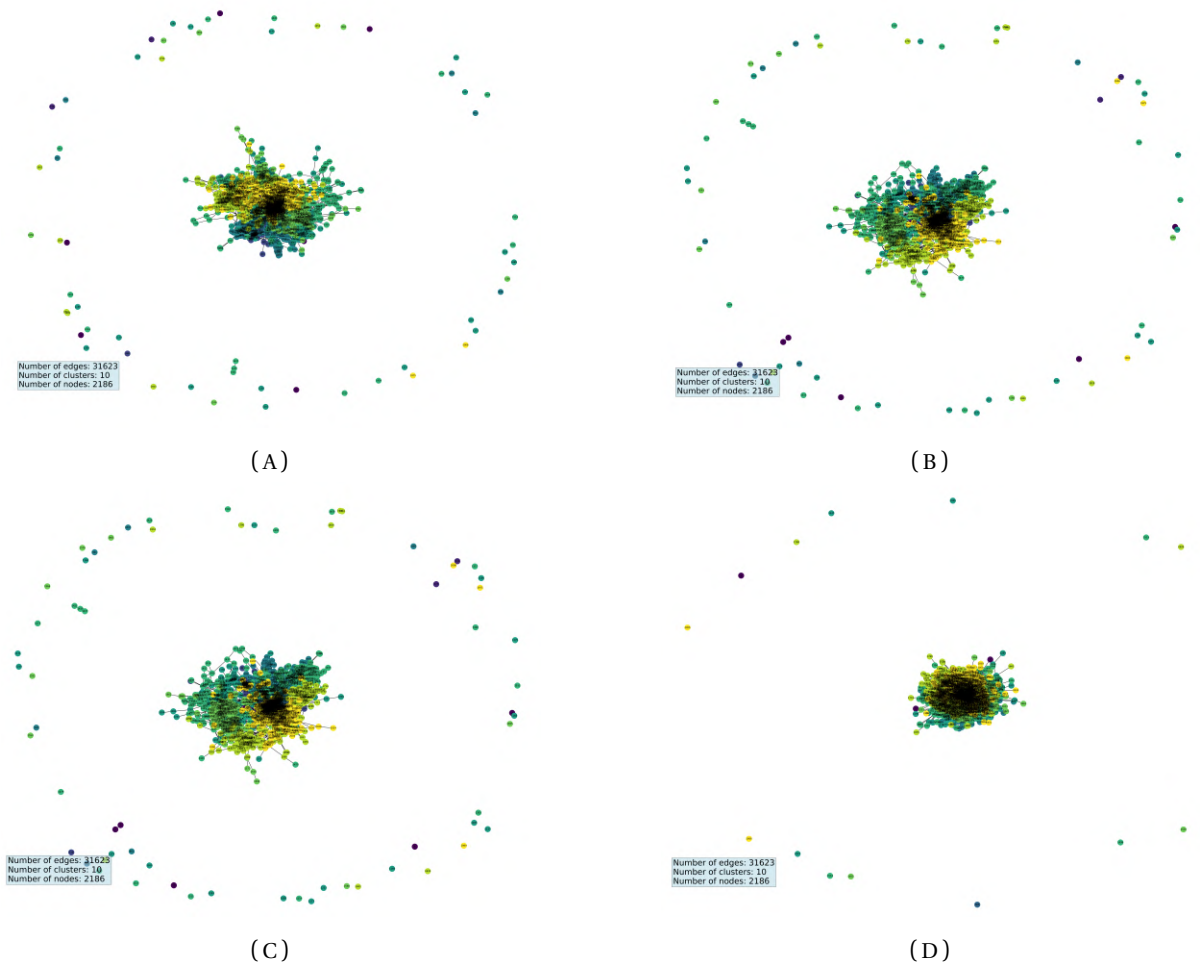


FIGURE 3.17: Visualization of the inferred cluster at convergence of HGNN. (A)Inferred cluster using BCE loss. (B) Inferred cluster using SMSE loss. (C) Inferred cluster using clustering loss. (D) Inferred cluster using cluster penalty loss

Figure 3.17 gives a visualization of the inferred bottom layer graph structure under each loss component. The node coloring is based on true cluster labels. We observe that for all loss components, the true cluster labels have been placed in single or small classes.

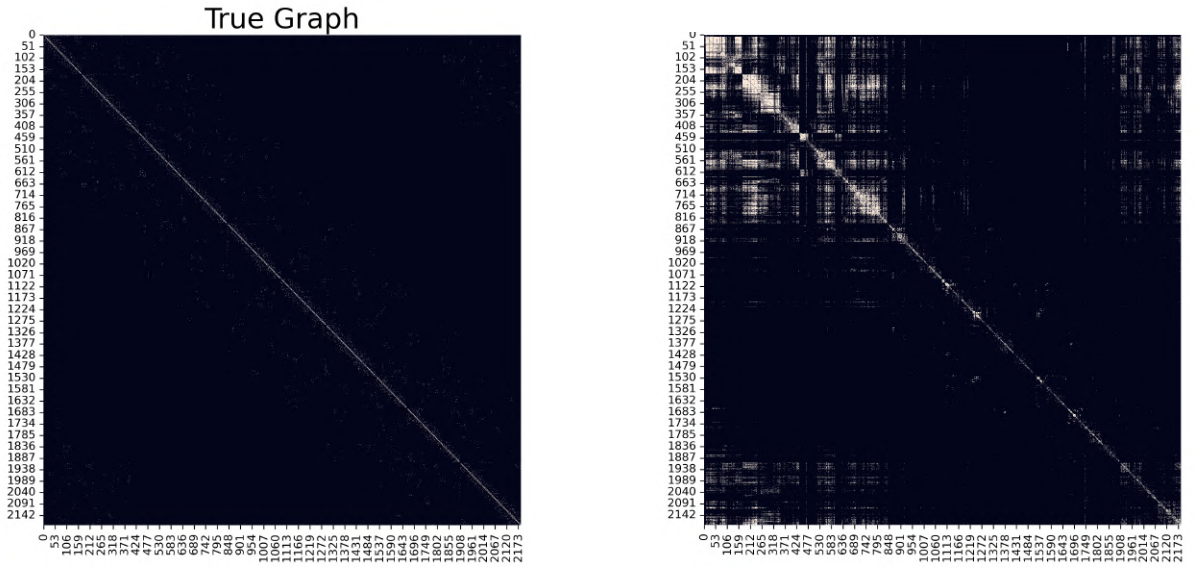
Loss Function Parameters				Clustering Performance			Inferred Cluster Information			
BCE	SMSE	CL	K	Homogeneity	Completeness	NMI	Clusters	Super nodes	Super Edges	Super Layer
0.5612	0	0	0	0.5634	0.4926	0.5256	70	479	966	4
0	2.9248	0	0	0.5415	0.4843	0.5113	91	527	782	4
0	0	0.2501	0	0.5416	0.4842	0.5113	91	527	782	4
0	0	0	26	0.4754	0.46824	0.4751	26	272	1494	4

TABLE 3.3: Performance metric of HGNN, on our simple pseudo-gene expression dataset using a connected small-world three-layer hierarchy with standard deviation 0.1.

Table 3.3 shows the clustering performance metrics of HGNN on the pseudo-gene expression dataset generated from a connected small-world 3-layer hierarchy with a standard deviation of 0.1. It can be observed that HGNN performs fairly well in terms of clustering performance with homogeneity, completeness, and NMI ranging from 0.47 to 0.56. Similarly, the inferred cluster information produces a result in terms of inferring the number of clusters, number of super nodes, and number of super edges, ranging from 26 to 91, 272 to 527, and 782 to 1494 respectively. The number of inferred super-layers is 4. This result gives an idea of the potential of HGNN to infer underlying graph structure in the data.

3.3.2.2 *Connected small-world three-layer hierarchy (standard deviation = 0.5)*

For these results, we assess the effectiveness of HGNN on a complex pseudo-gene expression dataset with a standard deviation of 0.5.



(A) (B)

FIGURE 3.18: Binary heatmap of simulated network adjacency and correlation inferred adjacency matrix. Both true adjacency binary heatmap (A) and correlation adjacency binary heatmap (B) are sorted matrices based on the true cluster the nodes belong to

Figure 3.18a and Figure 3.18b show the binary heatmap of the adjacency matrix for the true bottom layer and the inferred absolute correlation adjacency, respectively, in our complex pseudo-gene expression dataset with standard deviation 0.5. As shown in Figure 3.18, the inferred adjacency matrix captures some significant portion of the underlying network structure compared to the true adjacency heatmap. Similar to the scenario of 0.1 standard deviation data, a comparable pattern is seen here. The correlation adjacency reveals a relatively high number of edges in comparison to the truth.

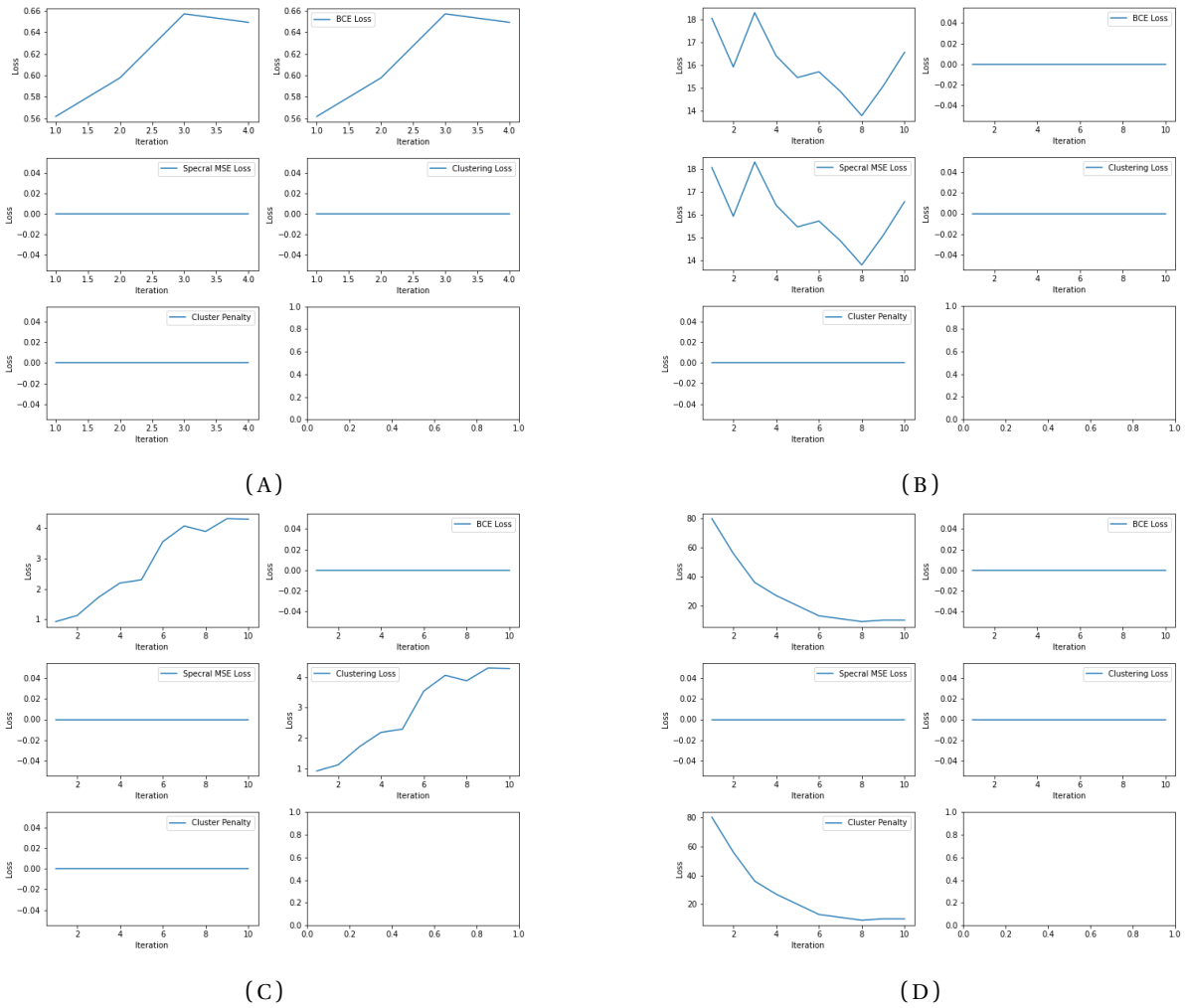


FIGURE 3.19: Hierarchy of 3 connected layers: Visualization of individual component of the loss function. (A) Binary Cross Entropy Loss curve, (B) Spectral Mean Square Error loss curve. (C) Clustering loss curve. (D) Penalty on the clustering curve.

In Figure 3.19, both BCE (Figure 3.19a) and clustering (Figure 3.19c) loss curves rise steadily, followed by a decrease, with the first iterations having the minimum loss value. On the other hand, the SMSE (Figure 3.19b) loss curve has an overall decreasing nature, with the minimum loss value at the eighth iteration. The penalty on the number of inferred clusters in the Figure 3.19d loss curve decreases over iterations, with the minimum loss value at the eighth iteration.

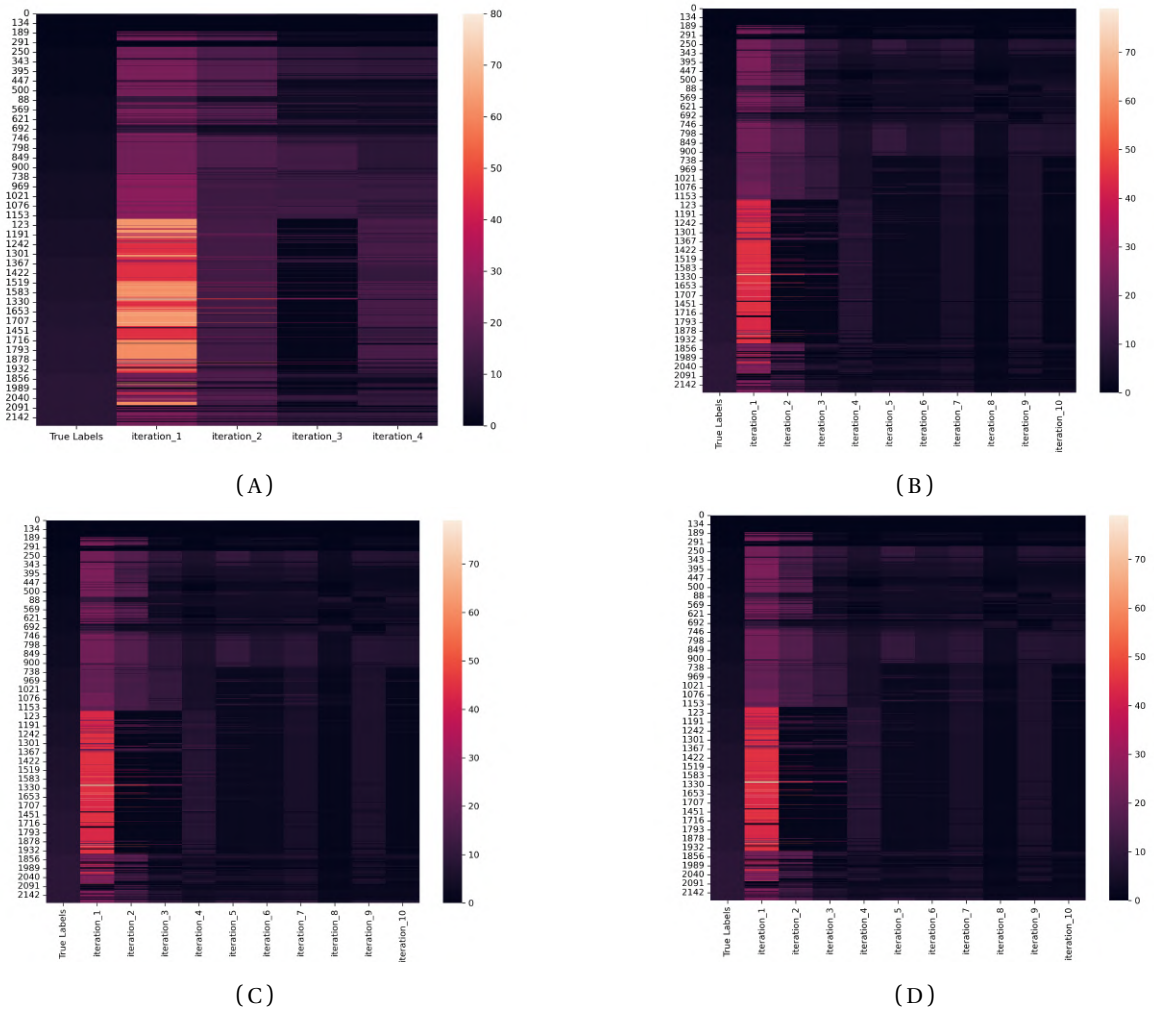


FIGURE 3.20: Visualization of inferred cluster labels and true cluster labels over iterations. (A) Cluster evolution using BCE loss. (B) Cluster evolution using SMSE loss. (C) Cluster evolution using Clustering loss. (D) Cluster evolution using cluster penalty loss.

Figure 3.20 shows the performance of HGNN on our complex pseudo-gene expression dataset, with a standard deviation of 0.5. The figure shows the true labels and inferred labels for each loss component at each iteration. We observe that the SMSE and penalty loss components inferred a reasonable number of clusters at their minimum loss values. However, the other loss components produced a larger number of inferred clusters compared to the truth.

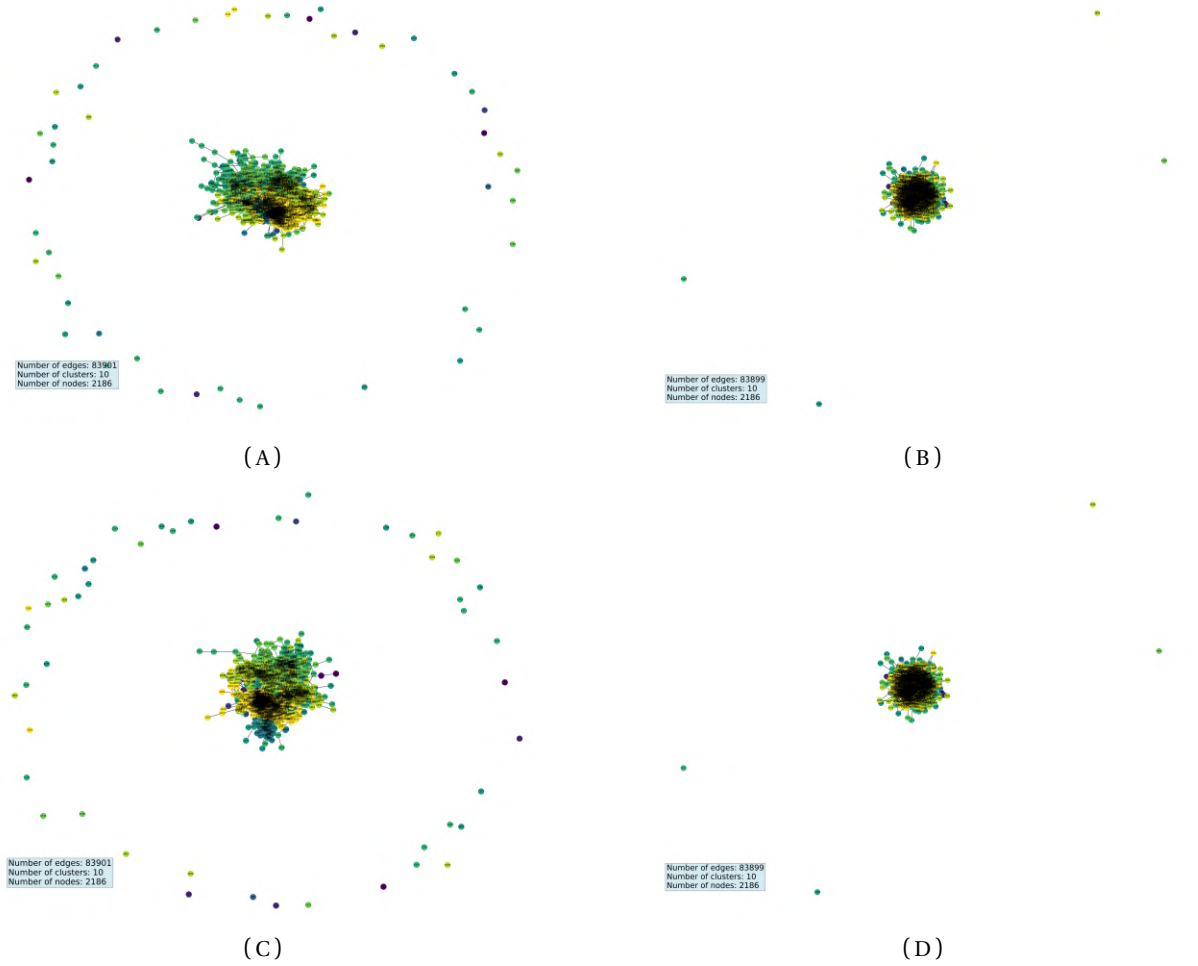


FIGURE 3.21: Visualization of the inferred cluster at convergence of HGNN. (A)Inferred cluster using BCE loss. (B) Inferred cluster using SMSE loss. (C) Inferred cluster using clustering loss. (D) Inferred cluster using cluster penalty loss

Figure 3.21 gives a visualization of the inferred bottom layer graph structure under each loss component with node coloring based on true cluster labels. We observe that for all loss components, the true cluster labels have been divided into single or smaller classes (similar to the observation in 3.17)

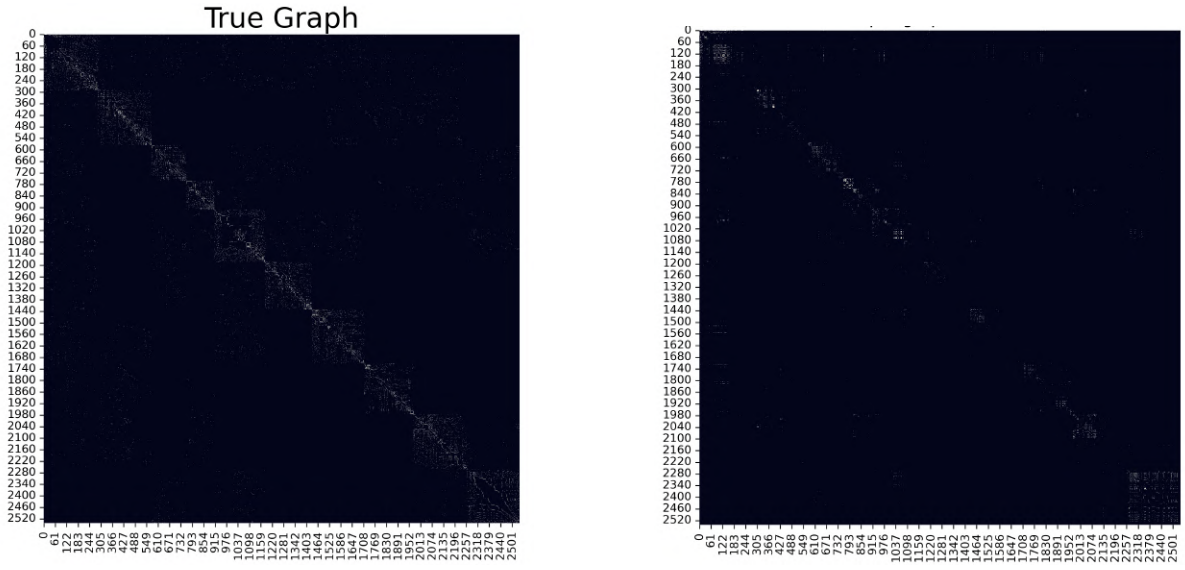
Loss Function Parameters				Clustering Performance			Inferred Cluster Information			
BCE	SMSE	CL	K	Homogeneity	Completeness	NMI	Clusters	Super nodes	Super Edges	Super Layer
0.5619	0	0	0	0.44	0.5242	0.4784	81	380	512	3
0	13.801	0	0	0.3101	0.5291	0.4	9	174	1379	3
0	0	0.9251	0	0.4421	0.5228	0.4791	80	380	512	3
0	0	0	9	0.31	0.5291	0.4	9	174	1379	3

TABLE 3.4: Performance metric of HGNN, on our simple pseudo-gene expression dataset using connected small-world three-layer hierarchy with standard deviation 0.5.

Table 3.4 shows the clustering performance metrics of HGNN on the pseudo-gene expression dataset generated from a connected small-world 3-layer hierarchy with a standard deviation of 0.5. We observe that HGNN performance in terms of clustering performance with homogeneity, completeness, and NMI ranges from 0.31 to 0.44, 0.52 to 0.53, and 0.4 to 0.48 receptively. Similarly, the inferred cluster information produces a result in terms of inferring the number of clusters, number of super nodes, and number of super edges, ranging from 9 to 81, 174 to 380, and 512 to 1379 respectively. The number of inferred super-layers is 4. This also gives an idea of the potential of HGNN to infer underlying graph structure in the data.

3.3.2.3 *Connected scale-free three-layer hierarchy (standard deviation = 0.1)*

We present the results of evaluating the performance of HGNN on our complex pseudo-gene expression dataset, which has a scale-free network structure and standard deviations of 0.1.



(A) (B)

FIGURE 3.22: Binary heatmap of simulated network adjacency and correlation inferred adjacency matrix. Both true adjacency binary heatmap (A) and correlation adjacency binary heatmap (B) are sorted matrices based on the true cluster the nodes belong to

Figure 3.22a and Figure 3.22b show the binary heatmap of the adjacency matrix for the true bottom layer and the inferred absolute correlation adjacency, respectively, in our complex pseudo-gene expression dataset with standard deviation 0.1. As depicted, the findings are consistent with previous scenarios. But here, the number of inferred edges in the correlation adjacency seems to be smaller when compared to the truth.

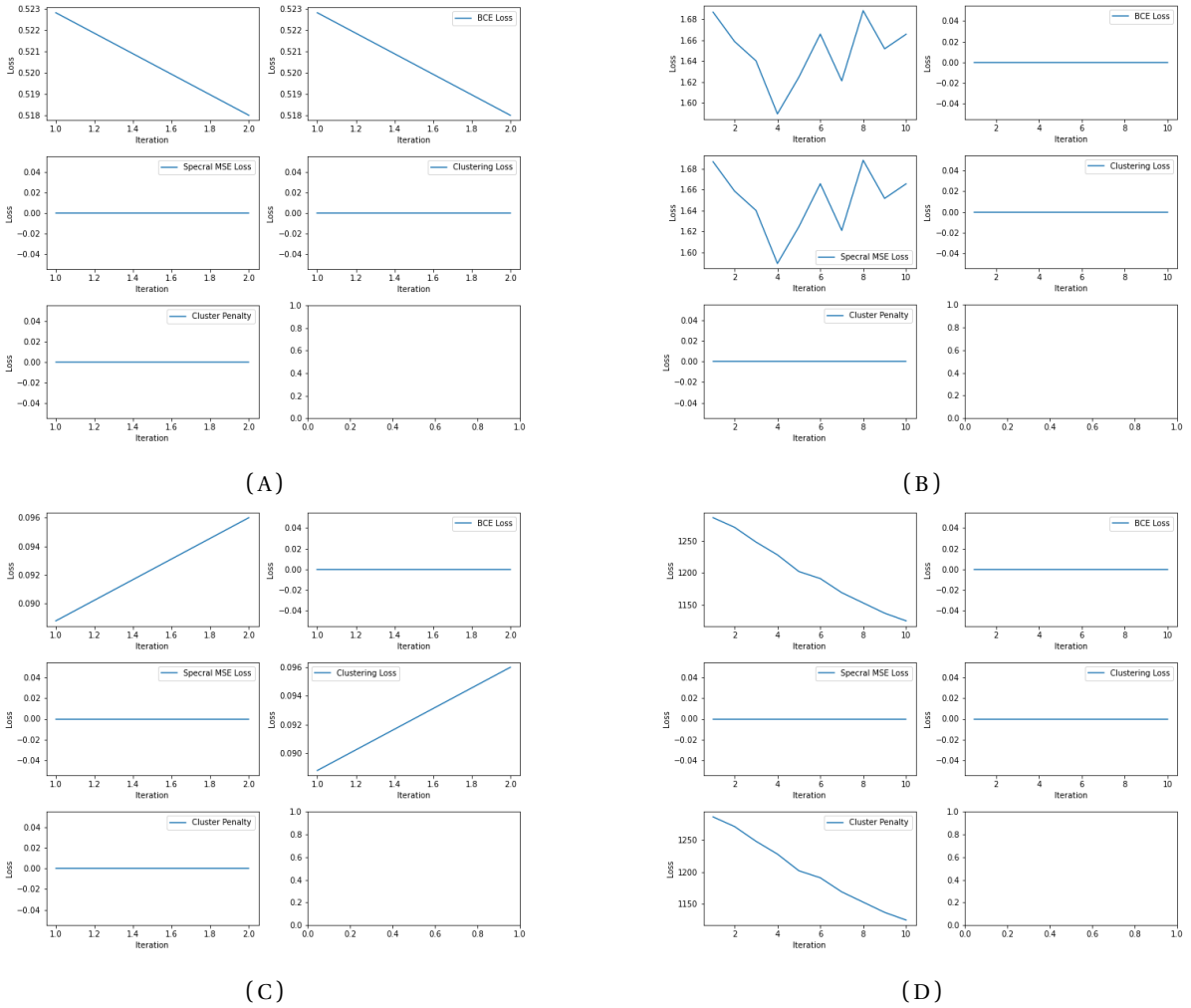


FIGURE 3.23: Hierarchy of 3 connected layers: Visualization of individual component of the loss function. (A) Binary Cross Entropy Loss curve. (B) Spectral Mean Square Error loss curve. (C) Clustering loss curve. (D) Penalty on the clustering curve.

In Figure 3.23, the visualization of the individual loss components of HGNN in our complex scale-free pseudo-gene expression dataset with standard deviation 0.1. It can be seen that the BCE (Figure 3.23a) and penalty (Figure 3.23d) loss curves show a decreasing trend, with the second and tenth iterations having the minimum loss values, respectively. In contrast, the SMSE (Figure 3.23b) and clustering (Figure 3.23c) loss curves generally show an

increasing trend, with the minimum loss values at the fourth and first iterations, respectively.

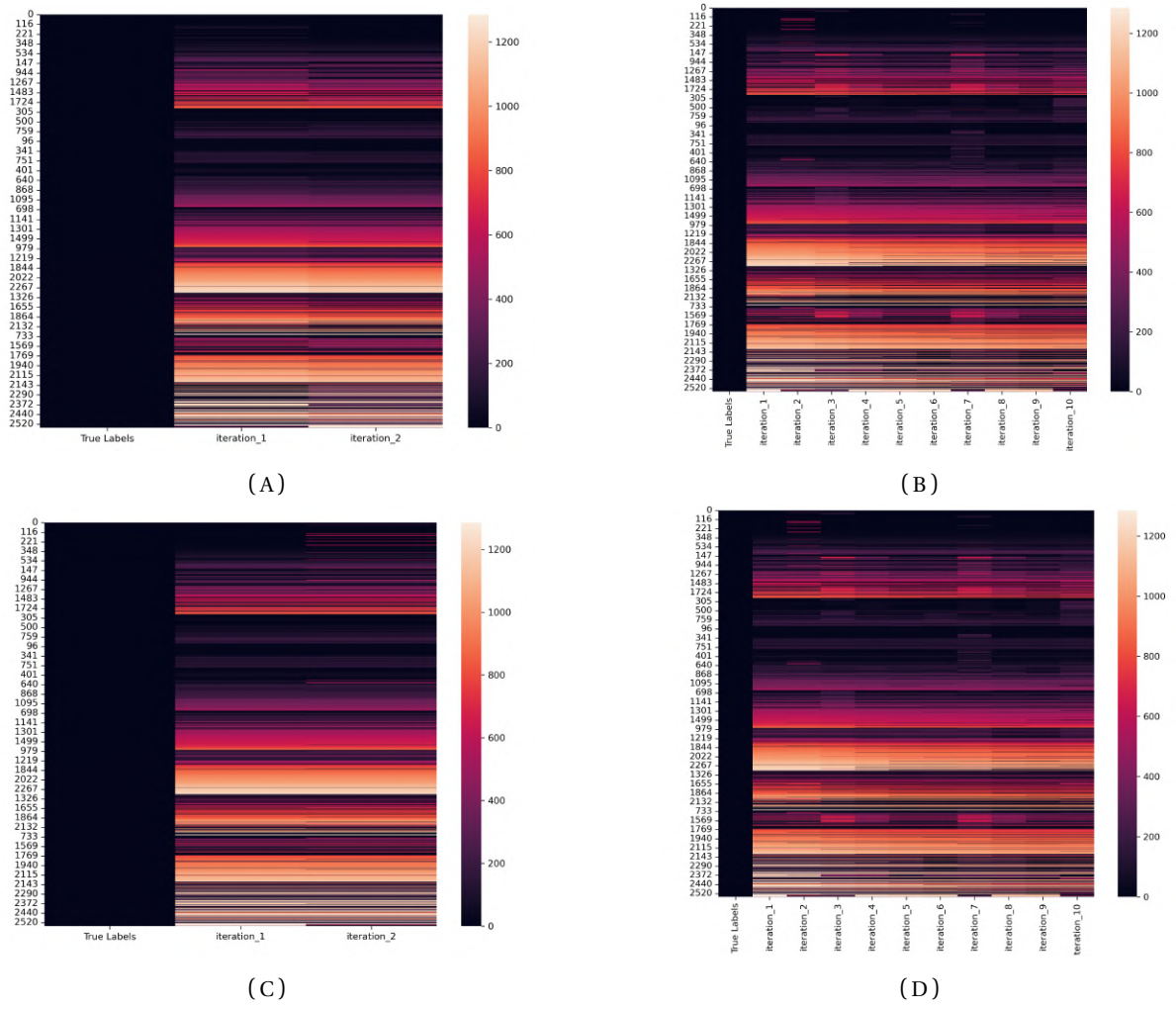


FIGURE 3.24: Visualization of inferred cluster labels and true cluster labels over iterations. (A) Cluster evolution using BCE loss. (B) Cluster evolution using SMSE loss. (C) Cluster evolution using Clustering loss. (D) Cluster evolution using cluster penalty loss.

In Figure 3.24, we see the true labels alongside the inferred labels for each individual loss component at each iteration. For all the individual components of the loss function, true cluster labels are placed into smaller clusters, resulting in many inferred clusters as observed in the small-world complex data set.

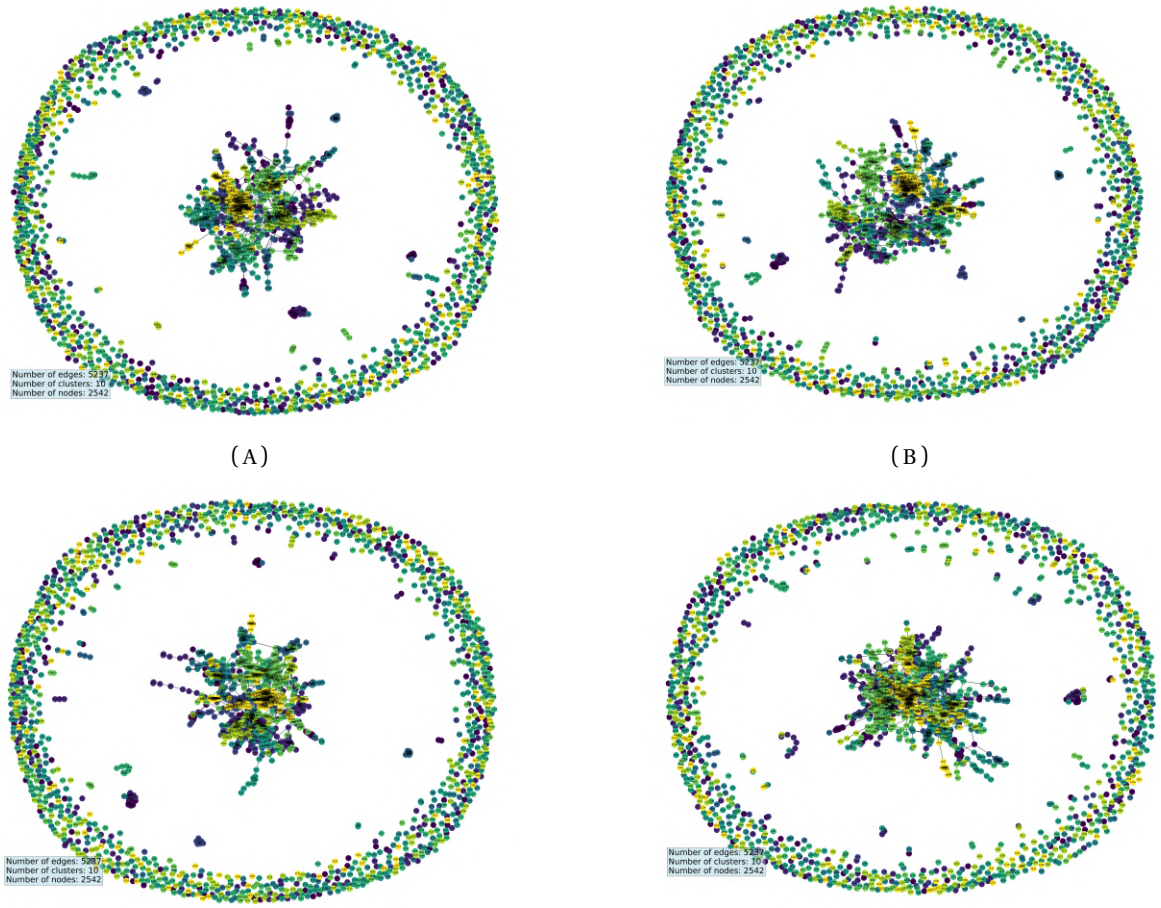


FIGURE 3.25: Visualization of the inferred cluster at convergence of HGNN. (A)Inferred cluster using BCE loss. (B) Inferred cluster using SMSE loss. (C) Inferred cluster using clustering loss. (D) Inferred cluster using cluster penalty loss

In Figure 3.25, the inferred bottom layer graph structure under each loss component is visualized with node coloring based on true cluster labels. It can be observed that for all components, the true cluster labels have been divided into smaller classes or combined with other class labels.

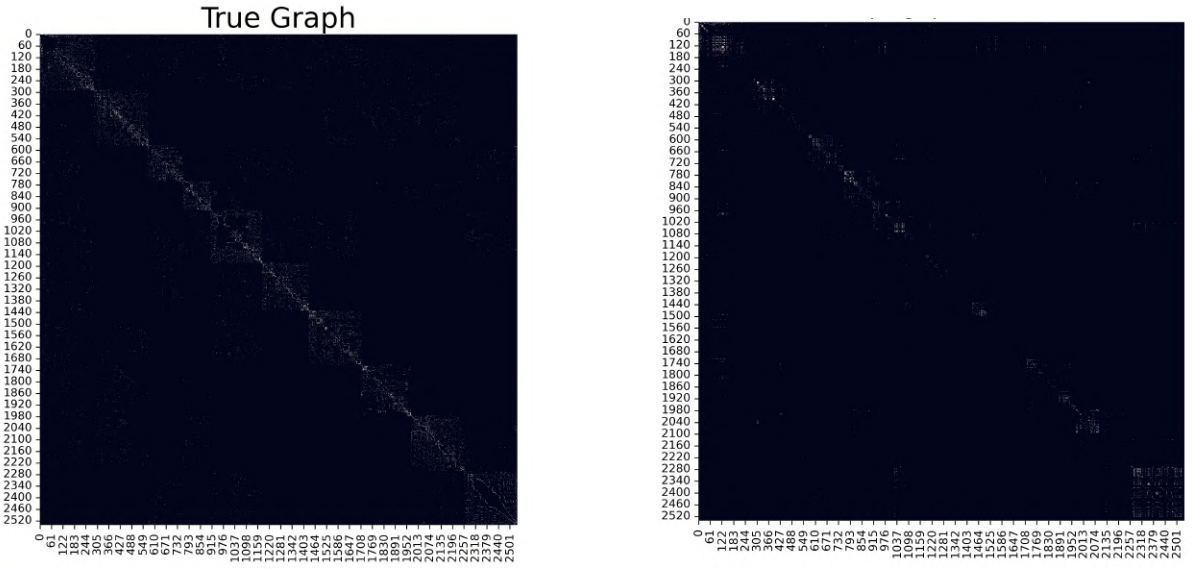
Loss Function Parameters				Clustering Performance			Inferred Cluster Information			
BCE	SMSE	CL	K	Homogeneity	Completeness	NMI	Clusters	Super nodes	Super Edges	Super Layer
0.518	0	0	0	0.8141	0.3201	0.4595	1249	5210	399	4
0	1.5893	0	0	0.7837	0.3137	0.448	1202	5064	458	4
0	0	0.0888	0	0.8349	0.3262	0.4691	1267	5271	357	4
0	0	0	1125	0.7385	0.2972	0.4221	1103	3557	481	3

TABLE 3.5: Performance metric of HGNN, on our simple pseudo-gene expression dataset using a connected scale-free three-layer hierarchy with standard deviation 0.1.

Table 3.5 shows the clustering performance metrics of HGNN on the pseudo-gene expression dataset generated from a connected scale-free three-layer hierarchy with a standard deviation of 0.1. We observe that HGNN performance in terms of clustering performance with homogeneity, completeness, and NMI ranges from 0.73 to 0.84, 0.3 to 0.33, and 0.42 to 0.47 receptively. Similarly, the inferred cluster information produces a result in terms of inferring the number of clusters, number of super nodes, and number of super edges, ranging from 1103 to 1249, 3557 to 5271 and 357 to 481 respectively with number of inferred super-layers ranging from 3 to 4.

3.3.2.4 *Connected scale-free three-layer hierarchy (standard deviation = 0.5)*

For this scenario, we assess the effectiveness of HGNN on our scale-free complex pseudo-gene expression dataset with a standard deviation of 0.5. The implementation entails setting the maximum number of iterations to 10 and performing 50 within-epoch loops with the absolute correlation threshold set to 0.5 to determine the inferred correlation adjacency matrix.



(A) (B)

FIGURE 3.26: Binary heatmap of simulated network adjacency and correlation inferred adjacency matrix. Both true adjacency binary heatmap (A) and correlation adjacency binary heatmap (B) are sorted matrices based on the true cluster the nodes belong to

Figure 3.26a and Figure 3.26b show the binary heatmap of the adjacency matrix for the true bottom layer and the inferred absolute correlation adjacency, respectively, in our complex pseudo-gene expression dataset with standard deviation 0.5. As shown in Figure 3.18, although the inferred adjacency matrix captures some significant portion of the underlying network structure compared to the true adjacency heatmap. Additionally, fewer edges seem to be inferred relative to the number of edges in the truth.

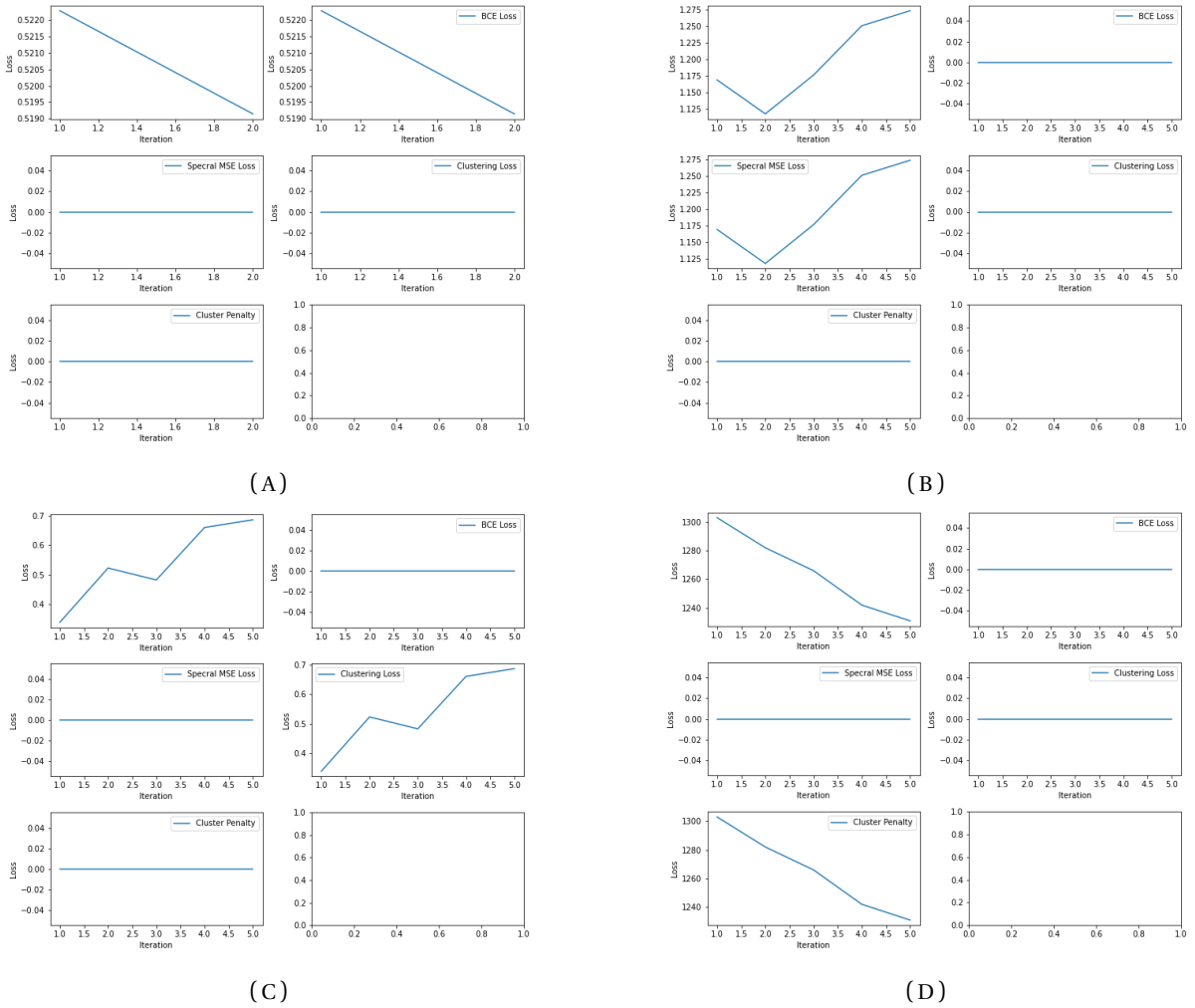


FIGURE 3.27: Hierarchy of 3 connected layers: Visualization of individual component of the loss function. (A) Binary Cross Entropy Loss curve, (B) Spectral Mean Square Error loss curve. (C) Clustering loss curve. (D) Penalty on the clustering curve.

In Figure 3.27, both SMSE (Figure 3.27b) and clustering (Figure 3.27c) loss curves rise steadily with second and first iterations respectively producing the minimum loss value. On the other hand, the BCE Figure 3.27a and penalty on the number of inferred clusters in the Figure 3.27d loss curves decrease over iterations with the minimum loss value at the second and fifth iteration respectively.

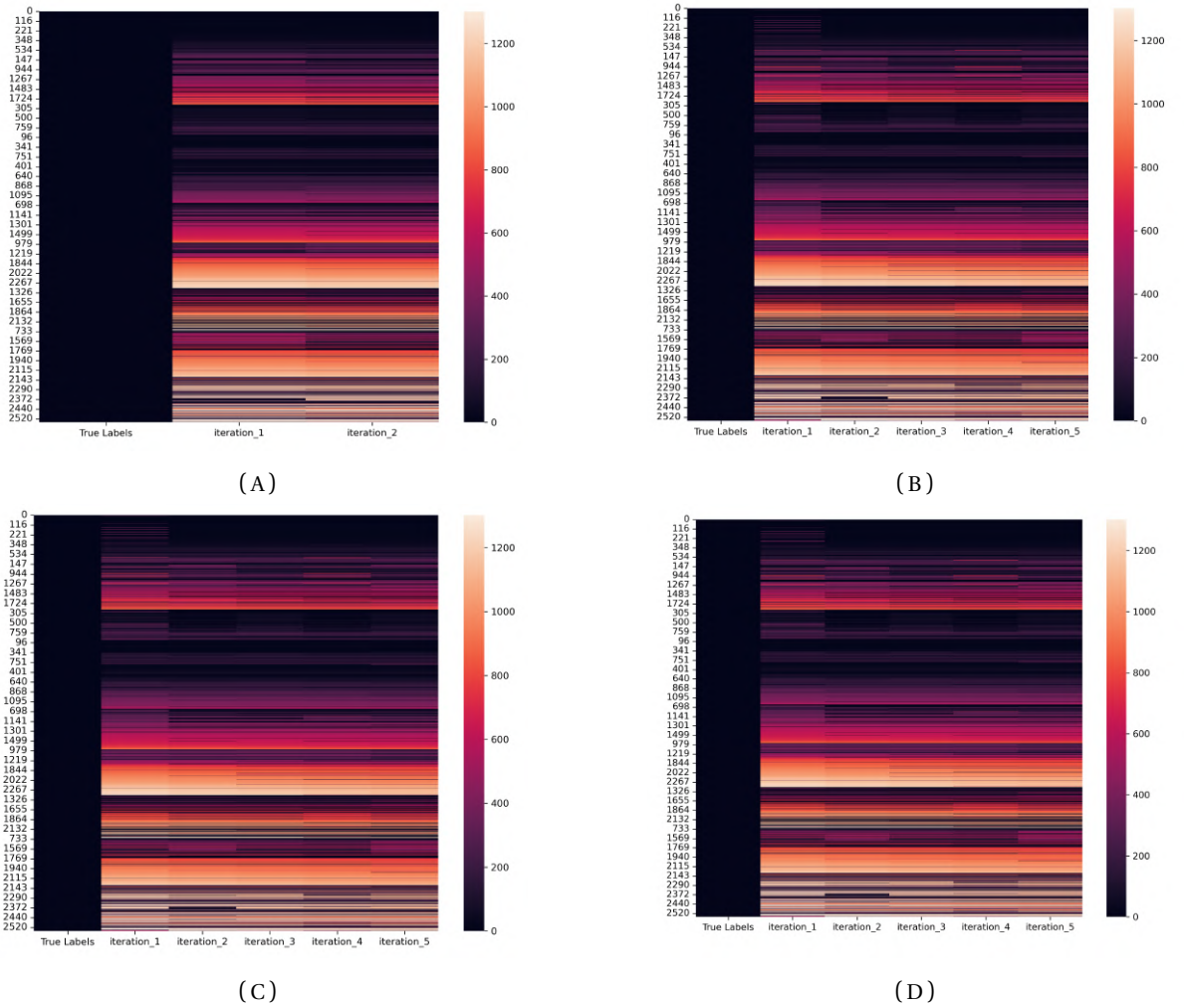


FIGURE 3.28: Visualization of inferred cluster labels and true cluster labels over iterations. (A) Cluster evolution using BCE loss. (B) Cluster evolution using SMSE loss. (C) Cluster evolution using Clustering loss. (D) Cluster evolution using cluster penalty loss.

Figure 3.28 shows the performance of HGNN on our complex pseudo-gene expression dataset, with a standard deviation of 0.5. The figure shows the true labels and inferred labels for each loss component at each iteration. We observe a similar result in Figure 3.24 where all the individual components of the loss function, true cluster labels are placed into smaller clusters, resulting in many inferred clusters as observed in the small-world complex data set.

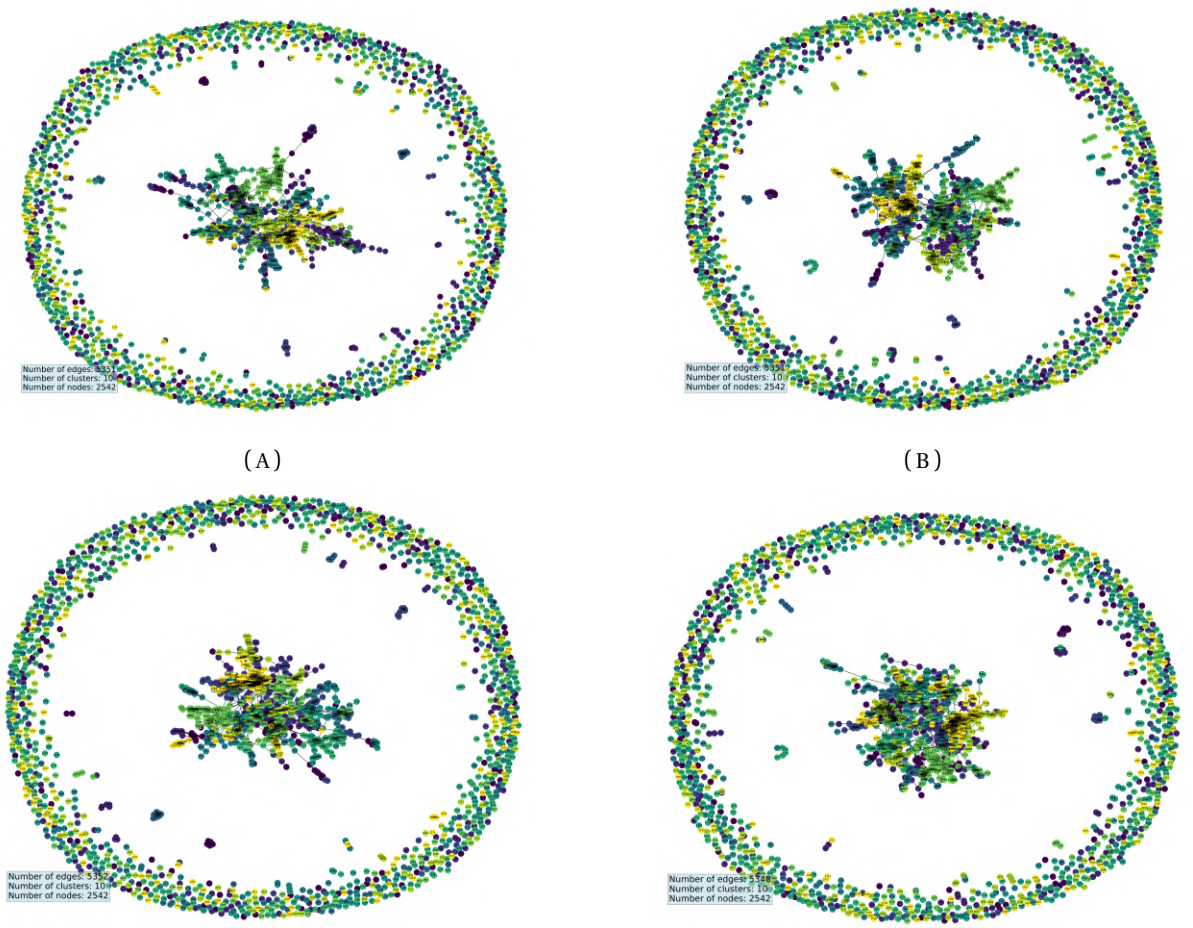


FIGURE 3.29: Visualization of the inferred cluster at convergence of HGNN. (A)Inferred cluster using BCE loss. (B) Inferred cluster using SMSE loss. (C) Inferred cluster using clustering loss. (D) Inferred cluster using cluster penalty loss

Figure 3.29 gives a visualization of the inferred bottom layer graph structure under each loss component with node coloring based on true cluster labels. It can be observed that for all components of the loss, the inferred network puts together nodes from different clusters in the true top layer or these nodes are split into smaller clusters.

Loss Function Parameters				Clustering Performance			Inferred Cluster Information			
BCE	SMSE	CL	K	Homogeneity	Completeness	NMI	Clusters	Super nodes	Super Edges	Super Layer
0.5192	0	0	0	0.8177	0.3214	0.4614	1252	5246	371	4
0	1.1178	0	0	0.8251	0.322	0.4635	1262	5270	387	4
0	0	0.3394	0	0.8312	0.3231	0.4653	1277	4026	310	3
0	0	0	1231	0.7888	0.3129	0.448	1209	5058	447	4

TABLE 3.6: Performance metric of HGNN, on our simple pseudo-gene expression dataset using connected scale-free three-layer hierarchy with standard deviation 0.5.

Table 3.6 shows the clustering performance metrics of HGNN on the pseudo-gene expression dataset generated from a connected scale-free three-layer hierarchy with a standard deviation of 0.5. We observe that HGNN performance in terms of clustering performance with homogeneity, completeness, and NMI ranges from 0.78 to 0.83, 0.31 to 0.33, and 0.44 to 0.47 receptively. Similarly, the inferred cluster information produces results in terms of inferring the number of clusters, the number of super nodes, and the number of super edges, ranging from 1209 to 1277, 4026 to 5270, and 310 to 447 respectively with a similar number of super-layers. This also gives an idea of the potential of HGNN to infer underlying graph structure in the data.

3.3.2.5 *Summary*

The results of the performance evaluation of HGNN on our complex pseudo-gene expression dataset generated from a connected three-layer hierarchy with the small-world and scale-free graph structure for standard deviations 0.1 and 0.5 indicate that the algorithm has the potential to infer the underlying hierarchical structure of the datasets. HGNN achieved fair clustering performance in terms of homogeneity, completeness, and normalized mutual information (NMI), ranging from 0.3 to 0.83, 0.3 to 0.56 and 0.4 to 0.47 respectively. In terms of homogeneity, we observed that all individual loss components resulted in better clustering performance on a scale-free graph structure compared to a small-world graph structure. However, in terms of the completeness metric, the small-world data performed slightly better than the scale-free data. Interestingly, for each dataset, the loss components

produced similar NMI scores. However, since the algorithm struggled to accurately determine the number of inferred super layers and nodes using the individual components of the loss function, it suggests a combination of the loss components may produce better results in terms of the evaluation metrics.

CHAPTER 4

CONCLUSIONS AND DISCUSSION

4.1 CONCLUSION

In our study, we introduced a new algorithm called HGNN, which is designed for learning highly dimensional GRNs. The algorithm consists of four key components. Firstly, we construct a graph by using an inferred adjacency matrix from the gene expression data to get an input graph. Secondly, we use Louvain community detection to infer the hidden upper layers of the input graph and build a message propagation pipeline for transferring and aggregating information from nodes and edges. Thirdly, we use the generated structure from the previous step and the gene expression dataset for learning embeddings of the expression dataset to infer a new graph. Finally, we perform structure updating and evaluation using homogeneity, completeness, and normalized mutual information performance metrics and update the structure by passing the inferred graph as the input if convergence is not met.

In conclusion, our study evaluated the effectiveness of HGNN by applying it to our pseudo-gene expression data containing both simple (dataset generated from a disconnected two-layer hierarchy) and complex (dataset generated from a connected three-layer hierarchy) datasets. The results demonstrated that HGNN is a powerful and effective method for inferring the underlying network structure in both simple and complex scenarios. In particular, we observed perfect clustering performance metrics and accurate inferred cluster information relative to the truth for the simple datasets. The complex datasets were more challenging, but HGNN was able to capture some key underlying structures in the true graph.

We also analyzed the individual components of the loss function and found that each component contributes significantly to the overall performance of HGNN. However, we note that the performance of HGNN could be further improved by tuning hyperparameters

such as the number of iterations, within-epoch loops, and absolute correlation threshold. Additionally, adopting a good loss component combination could also improve the performance of HGNN.

Overall, our findings suggest that HGNN is a robust and powerful tool for high dimensional network analysis, particularly for GRNs and other networks with similar properties. Future research could explore the potential of HGNN and optimize its hyperparameters for even better performance.

4.2 DISCUSSION

In this discussion section, we will explore some of the key findings and consider the implications of these findings for future research. The results from HGNN have provided important insights into the structure and patterns within our pseudo-gene expression dataset.

One important aspect of our analysis was the use of an appropriate loss function to measure the dissimilarity between genes within and between clusters. It was found that the choice of loss function components and weights has a significant impact on the performance of our HGNN algorithm. This suggests that carefully adopting the right combination and weight for the four loss component can potentially improve the performance of HGNN.

Another important consideration in our analysis was the use of a correlation threshold to infer an adjacency matrix for the input graph for HGNN. We found that setting an appropriate threshold was critical to achieving accurate and meaningful results. In particular, we observed that a too-low correlation threshold led to inferring few edges, while a too-high correlation threshold resulted in inferring a lots of false edges, which may combine distinct clusters. This suggests the correlation threshold selection should be carefully done based on some possible known characteristics of the gene expression data.

Finally, HGNN seems to be highly biased to the input graph. Thus, we also considered alternative methods(k-nearest neighbors, partial correlation, and precision) for generating the input graph for our HGNN algorithm. We found that the choice of input graph gener-

ation method had a significant impact on the results, with each method exhibiting different strengths and weaknesses. For example, the k-nearest neighbors method tended to be highly driven by the k parameter. Thus, when k is set very small, larger clusters are broken into small clusters, and when k is set high, there is an aggregation of smaller clusters. This suggests the need to carefully explore and evaluate different methods for generating the input graph based on some possible known characteristics of the gene expression data and the direction of inference.

Overall, our analysis highlights the importance of carefully considering the various components and parameters of HGNN for analyzing simple and complex datasets. By choosing appropriate correlation thresholds or evaluating alternative input graph generation methods and exploring different combinations of the loss function components, HGNN has the potential to provide insights into the underlying structure of a gene-level dataset to help identify patterns and relationships.

REFERENCES

Allen J.D., Xie Y., Chen M., Girard L., and Xiao G. 2012. Comparing statistical methods for constructing large scale gene networks. PloS one 7:e29348.

Badsha M., Fu A.Q., *et al.* 2019. Learning causal biological networks with the principle of mendelian randomization. Frontiers in genetics page 460.

Blondel V.D., Guillaume J.L., Lambiotte R., and Lefebvre E. 2008. Fast unfolding of communities in large networks. Journal of statistical mechanics: theory and experiment 2008:P10008.

Chen S. and Mar J.C. 2018. Evaluating methods of inferring gene regulatory networks highlights their lack of performance for single cell gene expression data. BMC bioinformatics 19:1–21.

Emmert-Streib F., Dehmer M., and Haibe-Kains B. 2014. Gene regulatory networks and their applications: understanding biological and medical problems in terms of networks. Frontiers in cell and developmental biology 2:38.

Faith J.J., Hayete B., Thaden J.T., Mogno I., Wierzbowski J., Cottarel G., Kasif S., Collins J.J., and Gardner T.S. 2007. Large-scale mapping and validation of escherichia coli transcriptional regulation from a compendium of expression profiles. PLoS biology 5:e8.

Friedman N., Linial M., Nachman I., and Pe'er D. 2000. Using bayesian networks to analyze expression data. Journal of computational biology 7:601–620.

Ghosh S., Das N., Gonçalves T., Quaresma P., and Kundu M. 2018. The journey of graph kernels through two decades. Computer Science Review 27:88–111.

Haeupler B., Kavitha T., Mathew R., Sen S., and Tarjan R.E. 2012. Incremental cycle detection, topological ordering, and strong component maintenance. ACM Transactions on Algorithms (TALG) 8:1–33.

Hinton G.E. and Salakhutdinov R.R. 2006. Reducing the dimensionality of data with neural networks. science 313:504–507.

Huynh-Thu V.A., Irrthum A., Wehenkel L., and Geurts P. 2010. Inferring regulatory networks from expression data using tree-based methods. PloS one 5:e12776.

Kipf T.N. and Welling M. 2016. Semi-supervised classification with graph convolutional networks. arXiv preprint arXiv:1609.02907 .

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

Kriege N.M., Johansson F.D., and Morris C. 2020. A survey on graph kernels. Applied Network Science 5:1–42.

Krouk G., Lingeman J., Colon A.M., Coruzzi G., and Shasha D. 2013. Gene regulatory networks in plants: learning causality from time and perturbation. Genome biology 14:1–7.

Mikolov T., Karafiát M., Burget L., Cernockỳ J., and Khudanpur S. 2010. Recurrent neural network based language model. *In* Interspeech, vol. 2, pages 1045–1048, Makuhari.

Pan S., Hu R., Long G., Jiang J., Yao L., and Zhang C. 2018. Adversarially regularized graph autoencoder for graph embedding. arXiv preprint arXiv:1802.04407 .

Reiser P., Neubert M., Eberhard A., Torresi L., Zhou C., Shao C., Metni H., van Hoesel C., Schopmans H., Sommer T., *et al.* 2022. Graph neural networks for materials science and chemistry. Communications Materials 3:93.

Reiss D.J., Plaisier C.L., Wu W.J., Baliga N.S., and cBio Center. 2016. Cytoscape app: Mcode. Methods in molecular biology 1343:339–352.

Rosenberg A. and Hirschberg J. 2007. V-measure: A conditional entropy-based external cluster evaluation measure. *In* Proceedings of the 2007 joint conference on empirical methods in natural language processing and computational natural language learning (EMNLP-CoNLL), pages 410–420.

Scarselli F., Gori M., Tsoi A.C., Hagenbuchner M., and Monfardini G. 2008. The graph neural network model. IEEE transactions on neural networks 20:61–80.

Schafer J. and Strimmer K. 2005. A shrinkage approach to large-scale covariance matrix estimation and implications for functional genomics. Statistical applications in genetics and molecular biology 4.

Shannon C.E. and Weaver W. 1949. The Mathematical Theory of Communication. University of Illinois Press, Urbana, IL.

Shi W. and Rajkumar R. 2020. Point-gnn: Graph neural network for 3d object detection in a point cloud. *In* Proceedings of the IEEE/CVF conference on computer vision and pattern recognition, pages 1711–1719.

Singh K.P., Miaskowski C., Dhruva A.A., Flowers E., and Kober K.M. 2018. Mechanisms and measurement of changes in gene expression. Biological Research for Nursing 20:369–382.

Veličković P., Cucurull G., Casanova A., Romero A., Lio P., and Bengio Y. 2017. Graph attention networks. arXiv preprint arXiv:1710.10903 .

Vishwanathan S.V.N., Schraudolph N.N., Kondor R., and Borgwardt K.M. 2010. Graph kernels. Journal of Machine Learning Research 11:1201–1242.

Wang J., Ma A., Chang Y., Gong J., Jiang Y., Qi R., Wang C., Fu H., Ma Q., and Xu D. 2021. scgnn is a novel graph neural network framework for single-cell rna-seq analyses. Nature communications 12:1882.

Wills P. and Meyer F.G. 2020. Metrics for graph comparison: a practitioner's guide. Plos one 15:e0228728.

Wilson R.C. and Zhu P. 2008. A study of graph spectra for comparing graphs and trees. Pattern Recognition 41:2833–2841.

Zhong Z., Li C.T., and Pang J. 2023. Hierarchical message-passing graph neural networks. Data Mining and Knowledge Discovery 37:381–408.

Zitnik M., Agrawal M., and Leskovec J. 2018. Modeling polypharmacy side effects with graph convolutional networks. Bioinformatics 34:457–466.