Instituto Alberto Luiz Coimbra de Pós-Graduação e Pesquisa de Engenharia COPPE UFRJ

COMMUNITY DETECTION ON NODE-ATTRIBUTED NETWORKS USING SELF-LEARNING AND GRAPH NEURAL NETWORKS

Rodrigo de Sapienza Luna

Dissertação de Mestrado apresentada ao Programa de Pós-graduação em Engenharia de Sistemas e Computação, COPPE, da Universidade Federal do Rio de Janeiro, como parte dos requisitos necessários à obtenção do título de Mestre em Engenharia de Sistemas e Computação.

Orientador: Daniel Ratton Figueiredo

Rio de Janeiro
Julho de 2023

COMMUNITY DETECTION ON NODE-ATTRIBUTED NETWORKS USING SELF-LEARNING AND GRAPH NEURAL NETWORKS

Rodrigo de Sapienza Luna

DISSERTAÇÃO SUBMETIDA AO CORPO DOCENTE DO INSTITUTO ALBERTO LUIZ COIMBRA DE PÓS-GRADUAÇÃO E PESQUISA DE ENGENHARIA DA UNIVERSIDADE FEDERAL DO RIO DE JANEIRO COMO PARTE DOS REQUISITOS NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE MESTRE EM CIÊNCIAS EM ENGENHARIA DE SISTEMAS E COMPUTAÇÃO.

Orientador: Daniel Ratton Figueiredo

RIO DE JANEIRO, RJ – BRASIL
JULHO DE 2023

Luna, Rodrigo de Sapienza

Community Detection on Node-Attributed Networks using Self-Learning and Graph Neural Networks/Rodrigo de Sapienza Luna. – Rio de Janeiro: UFRJ/COPPE, 2023.

XIII, 63 p.: il.; 29, 7cm.

Orientador: Daniel Ratton Figueiredo

Dissertação (mestrado) – UFRJ/COPPE/Programa de Engenharia de Sistemas e Computação, 2023.

Referências Bibliográficas: p. 57 – 63.

1. Graph Clustering. 2. Attributed Networks. 3. Graph Neural Networks. 4. Self Learning. I. Ratton Figueiredo, Daniel. II. Universidade Federal do Rio de Janeiro, COPPE, Programa de Engenharia de Sistemas e Computação. III. Título.

Ao meu amor que me
acompanhou em toda a minha
trajetória, Letícia Ecard.

Agradecimentos

Ao meu amor Letícia, que esteve ao meu lado em todos os momentos desde o nosso ensino médio e a gradução. Estar ao seu lado me fez amadurecer e sempre a ter forças para não desistir. Obrigado, meu amor!

Aos meu pais, Carlos e Sheila, e aos meus irmãos Gabriel e Rafael, obrigado pelo apoio.

Aos meus sogros, Adilson e Rogéria, e a minha cunhada Karine, pelo suporte e carinho ao longo desses anos.

Ao meu orientador, Daniel Ratton Figueiredo pelo suporte, paciência e dedicação para a relização desta dissertação. Além de ser um excepcional Professor e orientador, foi um grande amigo e conselheiro.

Aos Professores Pedro Olmo e Valmir Barbosa por participarem da banca de avaliação desta dissertação.

Aos Professores do Programa de Engenharia de Sistemas e Computação que tive o prazer de conhecer durante a pós-graduação.

Aos meus amigos e colegas do grupo de pesquisa do LAND, muito obrigado pelo apoio e companheirismo ao longo dessa jornada. Suas contribuições e amizade foram fundamentais para o meu crescimento e sucesso.

Agradeço também a todas as pessoas que de alguma forma contribuíram para a realização deste trabalho e para o meu desenvolvimento acadêmico e pessoal.

Por fim, agradeço a todos aqueles que acreditaram em mim e me incentivaram durante todo esse processo. Sem o apoio e encorajamento de vocês, nada disso seria possível.

Muito obrigado a todos!

Resumo da Dissertação apresentada à COPPE/UFRJ como parte dos requisitos necessários para a obtenção do grau de Mestre em Ciências (M.Sc.)

COMMUNITY DETECTION ON NODE-ATTRIBUTED NETWORKS USING SELF-LEARNING AND GRAPH NEURAL NETWORKS

Rodrigo de Sapienza Luna

Julho/2023

Orientador: Daniel Ratton Figueiredo

Programa: Engenharia de Sistemas e Computação

Agrupamento de grafos é um problema fundamental, pois encontra aplicações em uma infinidade de cenários diferentes. Enquanto métodos tradicionais têm se concentrado na estrutura da rede, uma variação recente considera o cenário em que os nós possuem atributos que também são informativos. Isso tem desencadeado novos métodos que aproveitam as informações da rede (arestas) e as informações dos nós (atributos) no desenvolvimento de novos algoritmos de agrupamento. Este trabalho propõe um novo framework que utiliza redes neurais de grafos (GNNs) em um processo de autoaprendizagem para resolver esse problema. Cada rodada de treinamento gera representações dos nós para o agrupamento no espaço euclidiano, influenciadas pelos resultados da rodada anterior. Além disso, um grafo de contexto é construído usando o grafo original para refinar ainda mais as representações dos nós. Resultados empíricos demonstram a eficácia de nossa abordagem na extração de informações tanto das arestas da rede quanto dos atributos dos nós em dados sintéticos. Ela supera algoritmos que se concentram exclusivamente na rede ou nos atributos quando as informações são limitadas. Além disso, várias rodadas de aprendizado superam consistentemente o treinamento de uma única rodada, oferecendo desempenho superior ao agrupamento clássico de grafos com GNN. Em conjuntos de dados reais, nossa metodologia mostra superioridade em relação aos métodos de ponta quando os tamanhos dos clusters são balanceados.

Abstract of Dissertation presented to COPPE/UFRJ as a partial fulfillment of the requirements for the degree of Master of Science (M.Sc.)

COMMUNITY DETECTION ON NODE-ATTRIBUTED NETWORKS USING SELF-LEARNING AND GRAPH NEURAL NETWORKS

Rodrigo de Sapienza Luna

July/2023

Advisor: Daniel Ratton Figueiredo

Department: Systems Engineering and Computer Science

Graph clustering is a fundamental problem as it finds applications in a myriad of different scenarios. While traditional methods have focused on network structure, a recent variation considers the scenario where nodes have attributes that are also informative. This has triggered novel methods that leverage network information (edges) and node information (attributed) in the design of novel clustering algorithms. This work propose a novel framework that utilizes graph neural networks (GNNs) in a self-learning process to this problem. Each round of training generates node representations for clustering in Euclidean space, influenced by the previous round's results. Additionally, a context graph is constructed using the original graph to further refine node representations. Empirical results demonstrate the efficacy of our approach in extracting information from both network edges and node attributes in synthetic data. It outperforms algorithms that solely focus on the network or attributes when information is limited. Furthermore, multiple rounds of learning consistently outperform single round training, offering superior performance to classic GNN graph clustering. In real datasets, our methodology shows superiority over state-of-the-art methods when cluster sizes are balanced.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

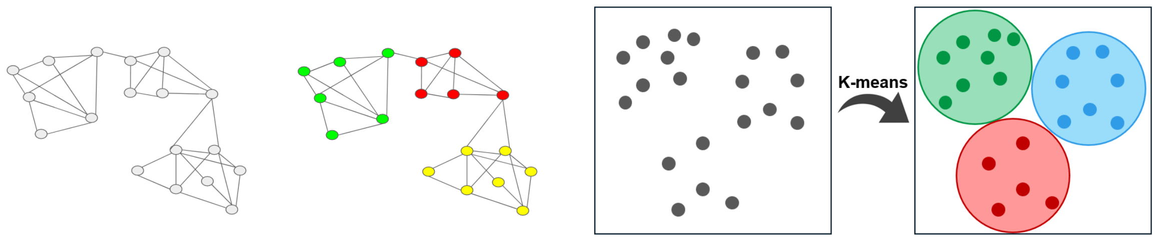
1.1 Motivation

Graph clustering, graph partitioning, or community detection is a fundamental problem in data science and network science, as depicted in Figure 1.1(a). It involves dividing the nodes of a graph into subsets that reflect some underlying structure, such as having more edges within subsets than between subsets.

A related, but even more fundamental problem is data clustering, as illustrated in Figure 1.1(b). It involves grouping similar data points together based on their intrinsic properties or similarities. Clustering techniques are widely applied in various fields, including pattern recognition, image analysis, customer segmentation, anomaly detection, and many others [8].

While graph clustering and data clustering have been studied independently for a long time, it is only recently that they have been unified into a single formulation. In attributed networks, each node in the network is associated with a feature that represents the node, such as the DNA of a person in a social network. These features can be represented as points in a space. In the context of attributed networks, nodes can be clustered using only the node attributes (data clustering), only the network structure (graph clustering), or a combination of both types of information (joint data and graph clustering). This work focuses on the problem of joint data and graph clustering, where both attribute information and network structure are considered together.

Integrating both node attributes and network structure to cluster nodes in an attributed network involves combining evidence from both sources. Intuitively, approaches that leverage both types of information should outperform methods that rely on only one type of information. The performance improvement is expected to be more significant when both attribute and network information are noisy and independent. In such cases, the joint approach can utilize more reliable attribute



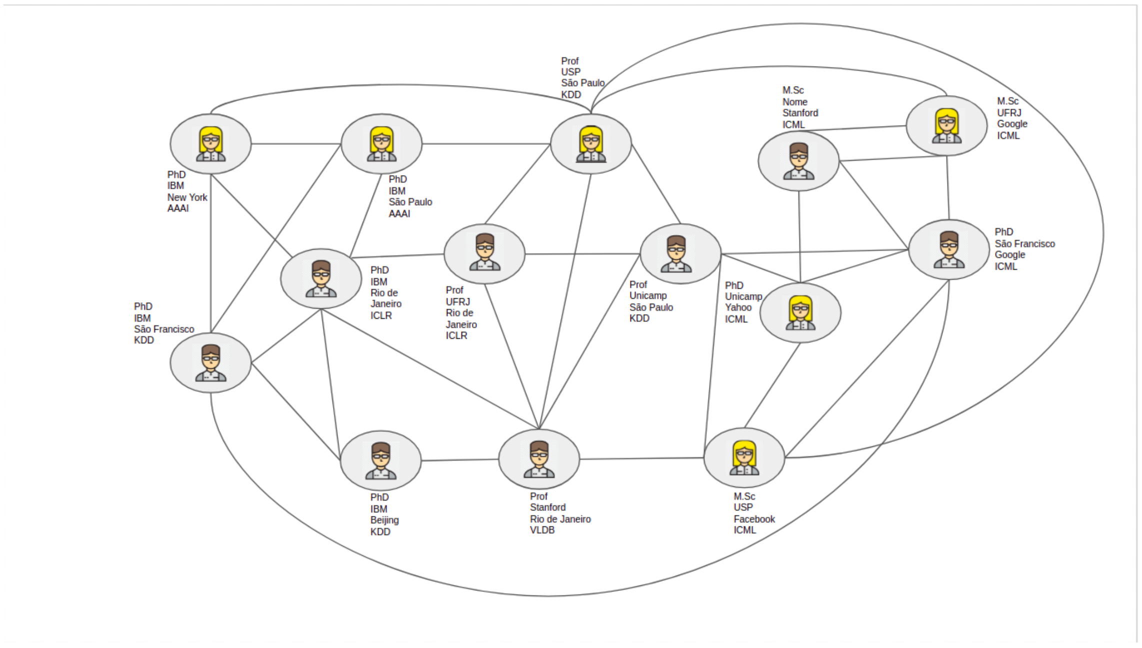
(a) Graphs (b) Data points

Figure 1.1: Methods to solve both of these problems, data point and graph clustering (community detection), have been investigated for decades.

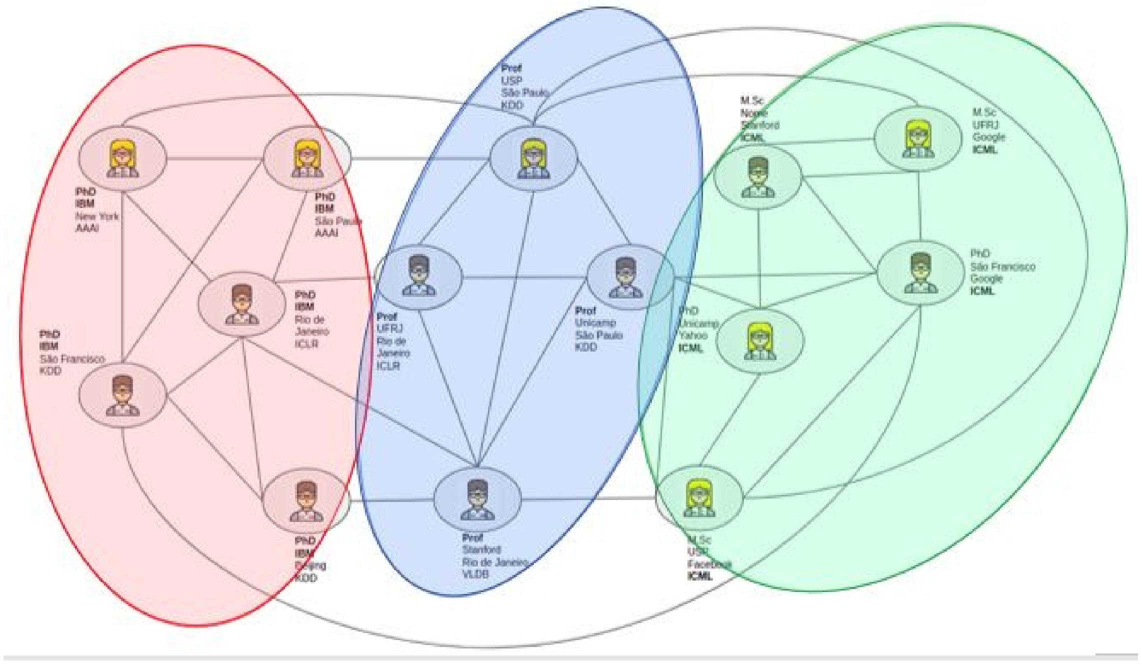
information to compensate for noisy network information, and vice versa. However, developing methodologies and algorithms that effectively leverage this intuition is a non-trivial task and has been the focus of active research in recent years. Figure 1.2 illustrates an attributed graph representing a collaborative research network. In this network, the nodes represent researchers who have collaborated on multiple papers, and the node attributes provide information about the researchers such as their qualifications, work locations, and journal reviewing activities. The clusters in the graph are formed by considering both the network structure and node attributes, as indicated by the different colors of circles.

Jointly using node attributes and network structure to cluster nodes of an attributed network requires mixing evidence from the attributes with evidence from the network structure. Intuitively, such approaches should outperform any approach that uses just one kind of information, and performance gains should be higher when both kinds of information are relatively noisy (and independent). In this scenario, the joint approach can use more reliable attribute information to correct for noisy network information, and vice-versa. Of course, designing methodologies and building algorithms that can leverage this intuition is not trivial, and has been an active area of research over the past few years [9–12].

An important consideration concerning graph or data clustering is prior information, or supervision. In a supervised setting, the labels of nodes or data points are available and indicate their clusters (or partially available in a semi-supervised setting). This information can be used to train a model that determines the cluster of unseen nodes. In contrast, in an unsupervised setting there is no prior information concerning the clusters. In modern and large problem instances, it is often the case that label information is not available (or is not reliable). This work considers the unsupervised scenario. The sole input to the problem here considered is a single instance of an attributed network.



(a) Attributed graph



(b) Clustered attributed graph.

Figure 1.2: Example of a node clustering algorithm that incorporates both network structure and node attributes for clustering, considering the attributes associated with each node.

1.2 Problem Formulation

Given an attributed graph undirected $G = (V, E, X)$, where $V = \{v_1, v_2, \ldots, v_n\}$ is a set of n nodes and E is a set of edges with $e_{ij} = (v_i, v_j) \in E$ if an edge exists between nodes v_i and v_j, and $X = \{x_1, x_2, \ldots, x_n\} \in R^{n \times F}$ is the attribute matrix, where each node v_i is associated to an attribute vector x_i with dimension F. The objective is to assign each node to a distinct and non-overlapping set C based on both the node structure and their attributes

This version of the problem considers the graph clustering (community detection) with the added aspect that attributes can be utilized to reveal the underlying communities. While many existing methods in network science focused solely on the network structure, this thesis aims to bridge the gap by considering both the attributes and the network structure for graph clustering.

GNNs have demonstrated successful applications in node classification and link prediction across different domains. It is not surprising that GNNs have also recently been employed for clustering nodes in attributed networks. In this context, representations learned by GNNs can serve as input for data clustering tasks. This thesis proposes a novel approach that combines GNNs and self-supervised learning techniques to tackle the problem of clustering.

1.3 Contribution

This thesis addresses the problem of clustering an attributed network in a fully unsupervised manner.
The main contributions are:

- Design and implementation of a framework using Graph Neural Networks in a self-supervised manner to addressing graph clustering.
- Novel neighborhood methodology which avoids under-reaching and over-smoothing commonly encountered in graph-based models.
- Empirical evaluation of the proposed framework on synthetic network and real networks. In order to gain a deeper understanding of the framework, we propose a method to create our own synthetic dataset to control the characteristics and properties of the data and analyze the influence of the network structure and attributes information on the learning process.

1.4 Structure

The remainder of the text is organized as follows:

- Chapter 2 presents a literature review on data and graph clustering (with and without attributes), methods for generating attributed graphs, systematic review of Graph Neural Networks, and the application of GNN in graph clustering. The most prominent frameworks and their fundamental aspects are discussed, including how researchers combine GNN models that are typically trained in a supervised manner to tackle graph clustering as an unsupervised task.

- Chapter 3 presents the proposed framework its characteristics and the motivation behind the mechanisms utilized. The chapter also describes how the model is trained in a self-learning manner, highlighting the iterative process.

- In Chapter 4, the proposed framework is empirically evaluated under different conditions, including varying network structures and attributes. Synthetic graphs were generated with the objective to examine individually the influence of network structure and attribute on the proposed framework. Additionally, the framework was evaluated using benchmarks commonly used for graph clustering. Results are reported in terms of the cluster quality found.

- Chapter 5 presents a conclusion of the work along with a path for future work.

Chapter 2

Background and related work

This chapter presents related work and important concepts, and it is organized as follows: Section 2.1 introduces data clustering, which is a classical problem in data science. Section 2.2 discusses classical community detection algorithms for unattributed graphs, such as the Girvan-Newman and Louvain algorithms. Section 2.3 introduces several generators of synthetic attributed community graphs. Finally, Section 2.4 focuses on related work that utilize Graph Neural Networks (GNNs) for community detection in attributed graphs.

2.1 Data Clustering

Clustering, as described by Jain et al. [13], is indeed one of the fundamental techniques in the field of Machine Learning. It is an unsupervised task that involves grouping similar data points based on their similarities or patterns in a high-dimensional space. The data points are represented as vectors in an n-dimensional Euclidean space, where each dimension corresponds to a feature or attribute. Most clustering algorithms seek to find natural clusters of data points based on their distances or similarities using a defined similarity function. Some commonly used similarity measures include Euclidean distance, cosine similarity, and correlation coefficients.

Clustering has major applications across a wide range of domains, including image segmentation, pattern recognition, anomaly detection, recommendation systems, social network analysis, and various other data science problems.

Jain et al. [13] categorizes clustering algorithms into two groups: hierarchical clustering algorithms and partitional algorithms. Hierarchical clustering is a method that build a hierarchy of clusters, the hierarchical clustering algorithms are derived from the single-link [14] and complete-link [15]. Partitional algorithms partition the data set into non-overlapping clusters. These algorithms optimize a specific objective function, such as minimizing the intra-cluster distance.

According to Jain [16], even though the k-means algorithm was proposed more than 50 years ago, it remains one of the most important partitional clustering algorithms. K-means aims to partition the data into K clusters by minimizing the sum squared distances between the data points and their respective cluster centroids. The objective function of the k-means algorithm can be represented as follows:

$$\min_{C} \sum_{i=1}^{k} \sum_{x \in C_i} ||x - \mu_i||^2 \tag{2.1}$$

where C represents the set of K clusters, x represents a data point, and μ_i represents the centroid of cluster C_i, where $|| \cdot ||$ represents the Euclidean distance between two points. The algorithm iteratively updates the cluster assignments and centroid positions until convergence is reached.

2.2 Network Community Detection

Graph clustering or graph partitioning or community detection involves partitioning the set of nodes of a graph into disjoint subsets that reflect some higher order structure (e.g., many more edges within the subsets than across subsets). Community detection is a fundamental problem in network science, with applications in social networks, biological networks, and information networks. Researchers from different fields have explored the problem of community detection for many years, employing various problem formulations and approaches, such as fast heuristics for minimizing modularity in network science or approximation algorithms for optimal ratio-cut in graph theory [17].

Traditional methods such as graph partitioning involve dividing the vertices into k groups of predefined sizes. The Kernighan-Lin algorithm, proposed by Kernighan and Lin [18], is a heuristic for graph partitioning, which partition the nodes into two disjoints subsets of equal (or nearly equal) size. Another popular technique is the spectral bisection method proposed by Barnes and Earl [19].

Note that most traditional algorithms for graph partitioning dot not perform well for community detection, because it is necessary to provide as input the number of groups and the group sizes. In community detection, the intention is to discover the community structure of a graph without prior knowledge about the community sizes. A large number of techniques have been proposed to find optimal communities in polynomial time. Most of these techniques are based on the optimization of objective function.

Modularity optimization is one of the most widely used techniques in community detection and the concept was proposed by Newman and Girvan [20]. The Modularity M, Equation 2.2, quantifies the community quality where A is the adjacency matrix, m is the number of edges and k_i and k_j, are respectively, the total degree of the nodes within their respective communities. It measures the difference between the actual number of edges within communities (groups) in a graph against the expected number of edges if the edges were randomly distributed according to a random graph or a null model. The Modularity M value ranges from -1 to 1, where a value closer to 1 indicates a strong community structure, while a value closer to 0 or negative values indicate a weak or no community structure.

$$M = \frac{1}{2m} \sum \left[A_{ij} - \frac{k_i \cdot k_j}{2m} \right] \tag{2.2}$$

As shown in Figure 2.1, the node coloring represents the community assignments. Note that in Figure 2.1(a) the maximum value accurately captures the two obvious communities, with a value of $M = 0.41$. Figure 2.1(b) has nodes associated with a non-obvious community. $M = 0$ (Figure 2.1(c)) indicates a network with only one community, and a negative value (Figure 2.1(d)) indicates that every node belongs to a different community.

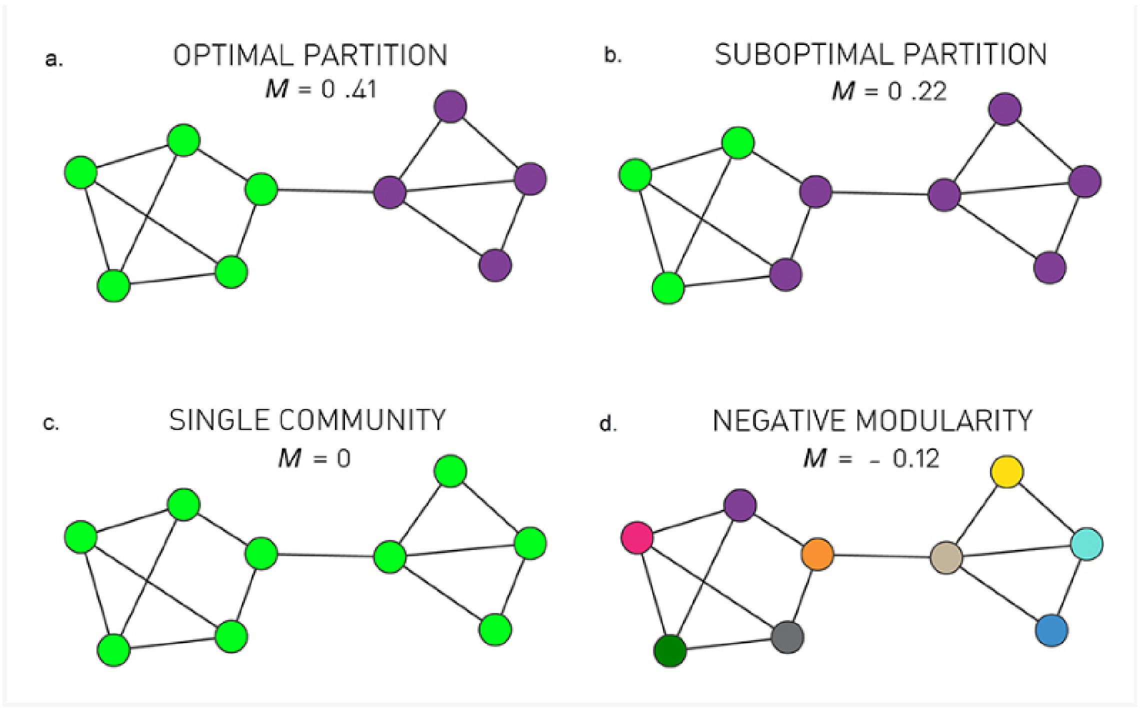


Figure 2.1: Different ways of assigning nodes to communities in the same graph, can have a significant impact on the modularity measure. Extract from [1]

The Newman and Girvan algorithm [20] was the first heuristic algorithm proposed to optimize the modularity quality measure. Another efficient approach for community detection in graphs is the Louvain algorithm proposed by Blondel et al. [21]. It is known for its scalability and ability to handle large-scale networks.

The Louvain algorithm consists of a two-phase iterative procedure. In the first phase, each node is assigned to its own community. Then, the algorithm iteratively optimizes the modularity by greedily moving nodes between communities with the objective of maximizing the modularity gain. In the second phase, the obtained communities are treated as individual nodes, and the same optimization process is applied to identify new communities. The Louvain algorithm performs well and is widely used due to its efficiency, scalability, and ability to detect communities.

Although these methods are efficient, they suffer from certain limitations. First, the algorithm fails to identify small or overlapping communities [22]. Second, they are strongly dependent on the initial partitioning [23]. Third, they fail on graphs that have communities with different sizes [24]. Indeed, Modularity algorithms are typically designed for static graphs, as they assume a fixed network structure without considering the addition or removal of nodes during runtime. When a new node is added, the algorithm must be re-executed to update the community assignments and reflect the changes in the network.

In addition to the challenge of handling communities with different sizes, another limitation of traditional community detection methods is that they often do not consider the attributes or features associated with the nodes in the graph. These methods primarily rely on the network's structural information, such as edge connections, to identify communities. However, in many real-world scenarios, the attributes or features of nodes can provide valuable information that can aid in community detection

2.3 Stochastic Block Model

Stochastic Block Model (SBM) [25] is a generative model proposed for modeling social networks. SBM produces graphs communities following the homophily concept, which refers to the tendency of individuals with similar characteristics be connected to each other. This phenomenon has been widely observed in various social, biological, network systems and collaborative networks.

SBM (Stochastic Block Model) groups N nodes into K blocks, with V_i representing the set of nodes in the i-th block. Each node is assigned to a specific block, and the probability of an edge existing between two nodes depends on the blocks they belong to. This is determined by a symmetric matrix $B \in R^{K \times K}$, which specifies the connectivity probabilities between the blocks (communities or clusters). The entry b_{ij} in this matrix represents the probability of an edge $e(u, v)$ existing between a node $u \in V_i$ and a node $v \in V_j$ in different blocks. In general, the probability of edges within the same cluster (intra-cluster) is higher than the probability of edges between different clusters (inter-cluster), denoted as $b_{ii} > b_{ij}$ for all $i \neq j$. This

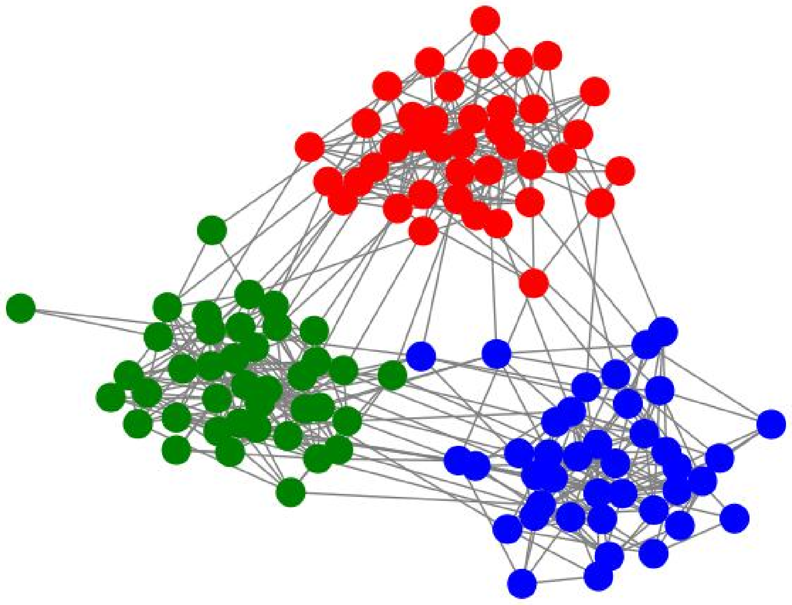


Figure 2.2: Stochastic block model with $V_i = 40$ and probability intra-cluster is 0.2 and inter-cluster 0.01.

reflects the concept of homophily.

For example, consider the SBM sample shown in Figure 2.2, where $|V_i| = 40$, $K = 3$, and the following probability matrix is used:

$$B = \begin{pmatrix} 0.2 & 0.01 & 0.01 \\ 0.01 & 0.2 & 0.01 \\ 0.01 & 0.01 & 0.2 \end{pmatrix} \tag{2.3}$$

2.3.1 Attributed Network

An attributed network nodes not only have connections (edges) but also possess attributes or features. These features can provide additional information about the nodes and their relationships, enabling a richer analysis of the network. In this kind of network, each node is associated with a set of attributes or features values. Features can be categorical or numerical, representing the characteristics of the node. For example, in a social network, nodes could have attribute as age, gender, occupation and interests and in a biological network nodes can have attribute as DNA information.

Recently, the popularity of machine learning on graph-structured data has been increasing, with various methods being proposed [26]. Tasks as node-classification, link prediction and graph clustering have been adopted in several application domains, such as recommender systems, social networks and web. Despite the abundance of graph learning methods, there is a limited number of real-world datasets available as benchmarks, with many of them derived from academic citation networks. Among these datasets: Cora, Citeseer and PubMed are the most commonly analyzed in various works [27–31].

Evaluating new algorithms on specific types of networks provides a limited sample to determine their overall performance and effectiveness. Therefore, it is crucial to have a large and diverse set of graph datasets to ensure that overfitting is avoided. In order to address this issue, a feature-rich synthetic graph generator can be employed to produce a variety of synthetic graphs with different characteristics and properties. This allows for a thorough examination of algorithms in various scenarios, facilitating a more comprehensive evaluation and understanding of their performance. Researchers can explore different network structures, sizes, community distributions, and other important factors that may impact the behavior of the algorithm. This approach helps to gain insights into the strengths, weaknesses, and generalizability of the algorithm across different types of graphs, leading to more robust and meaningful evaluations.

Several methods have been proposed for generating synthetic attributed graphs, and one of them is the Degree-Corrected Stochastic Block Model (DC-SBM) [32]. DC-SBM is a probabilistic model that extends the Stochastic Block Model (SBM) by incorporating continuous vector attributes. It uses a mixture of multivariate Gaussian distributions to generate the attribute vectors for the nodes in the graph. By combining the block structure of SBM with the continuous attributes, DC-SBM allows for the generation of synthetic graphs that capture both the network structure and attribute characteristics of real-world networks. Furthermore, DC-SBM takes into account the varying degrees of connectivity of nodes when modeling the network structure and attribute dependencies. This makes DC-SBM a flexible and realistic approach for generating synthetic attributed graphs for various research and evaluation purposes.

The generator GenCAT [33] proposes a user-flexible attributed graph generator that incorporates two phenomena: Core/Border nodes and Homophily/Heterophily. The Core/Border nodes phenomenon refers to the existence of distinct classes of nodes in real-world graphs. Core nodes have higher degrees, while Border nodes have lower degrees and act as connectors between the core nodes and the rest of the graph. The second phenomenon, homophily/heterophily, captures the tendency of nodes with similar attributes to be connected. Homophily refers to the preference for nodes with similar attributes to connect, whereas heterophily refers to the preference for nodes with different attributes to connect. This means that nodes with dissimilar characteristics are more likely to be connected. GenCAT incorporates homophily and heterophily by generating node attributes that influence the likelihood of connections between nodes. Therefore, GenCAT offers flexibility in generating attributed graphs with different levels of core/border node structure and homophily/heterophily. This allows for the generation of graphs that capture varying degrees of connectivity and attribute dependencies.

2.3.2 Simple Network Attibuted Generator

In our work, our primary focus is on developing a framework that can effectively capture the relationship between the graph structure and the associated attributes to detect meaningful communities within the network. We are interested in evaluating the performance of this framework and gaining a comprehensive understanding of its capabilities. It is crucial to consider the limitations that arise when isolating the signals from network topology and node attributes, as well as when combining both signals. By examining these scenarios, we can identify potential challenges that may affect the accuracy and effectiveness of the framework in capturing the complex relationship between these signals. This understanding will contribute to a more informed analysis of the framework's performance and guide future improvements.

We employed a simple approach to generate synthetic attributed graphs. We incorporated both graph connectivity and node attribute information to infer node-to-community assignments by assuming that they are conditionally independent, given the community membership label. In this way, we incorporate the Stochastic Block Model (SBM) to generate the graph structure and the nodes attributes was generated by a mixture of Gaussian Distributions depending on the community they belong to.

From the communities K sampled by the Stochastic Block Model (SBM), the vector attribute X_u of a node u, which belongs to community k, is sampled from the following distribution:

$$X_u \sim \mathcal{N}(\mu_k, \sigma_k^2 | K = k) \tag{2.4}$$

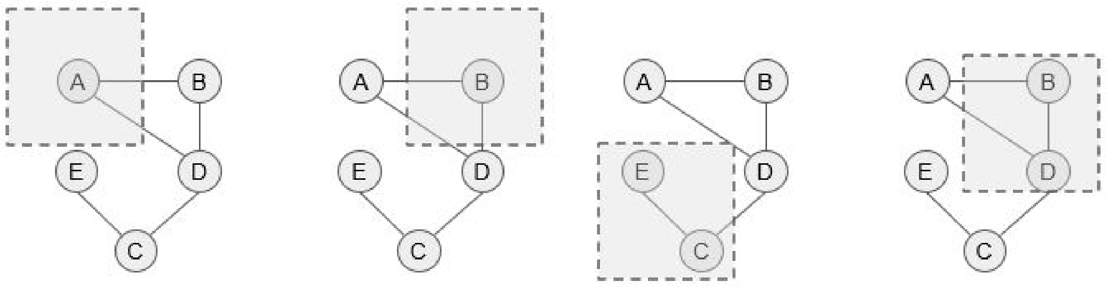
Note that to increase the attribute signal between nodes that belong to the same cluster, we can increase the distance between the Normal means and spread out the data attributes around the Normal mean with a smaller standard deviation

2.4 Graph Neural Network and Community Detection

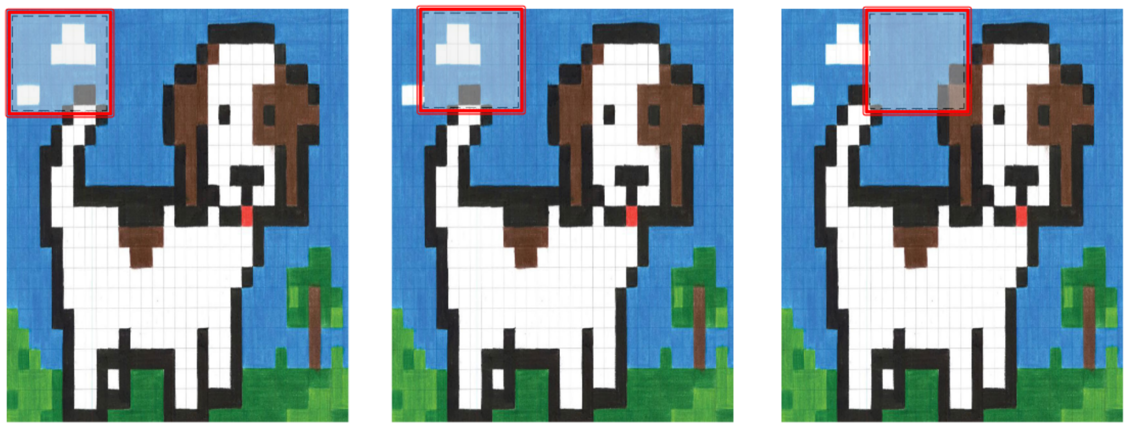
2.4.1 Overview

Graph neural networks (GNNs), also known as deep learning on graphs, graph representation learning, or geometric deep learning, have emerged as a hot research topic in machine learning with a wide range of applications across various domains [30, 34]. Unlike traditional deep learning models that operate on Euclidean structure data (e.g., images, sequence text), GNNs are specifically designed to handle non-Euclidean geometric data, represented as graphs, which are widely used to model relationships (edges) between objects (nodes) [35].

Unlike Euclidean data, the nature of graph-structured data implies lack properties such as global parameterization, vector space structure, shift invariance and a common system of coordinates. As a result, basic operations like convolution, which are well-defined and widely used in Euclidean spaces, are not easily applicable or even well-defined on non-Euclidean domains such as graphs.



(a) The convolutional operator is not directly applicable to graphs due to the absence of coordinates and the lack of permutation invariance among the nodes.



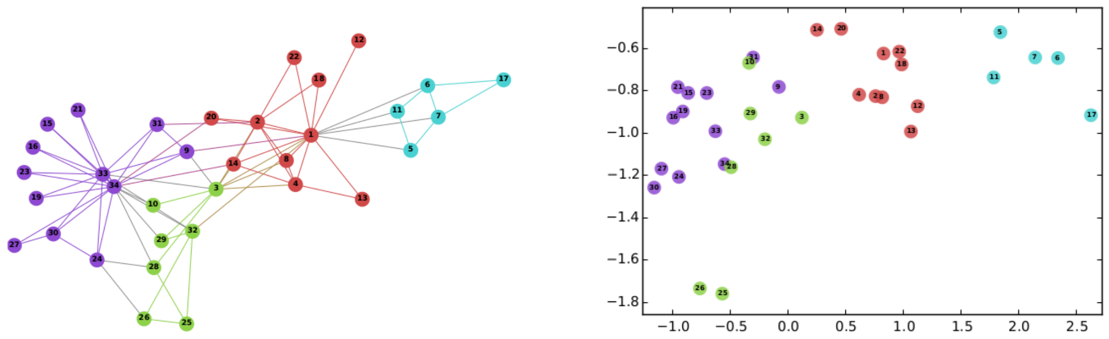
(b) Convolutional operator applied in a image

Figure 2.3: Convolutional operator applied in two kinds of data: image of a dog and a graph.

With the recent advancements in deep learning, particularly in convolutional neural networks (CNNs) [36], there have been efforts to extend CNNs to non-Euclidean domains, such as graph-structured data. These extensions aim to adapt the convolution operation to work effectively on graphs, taking into account the unique characteristics of graph data. CNN have the ability to extract multi-scale localized spatial features and compose from input data through the use of convolutional layers. These localized features are then combined and composed to construct highly expressive representations that capture the relevant information. The key mechanisms for the success of CNNs are local connections, weight sharing, and multiple layers. Although these mechanisms also are important for graph learning, CNNs can only be applied to Euclidean data. Due to the non-Euclidean nature of graphs, it becomes challenging to define the position of convolutional filters in the graph, as showed in the Figure 2.3, the convolution operator is represented by the red square kernel.

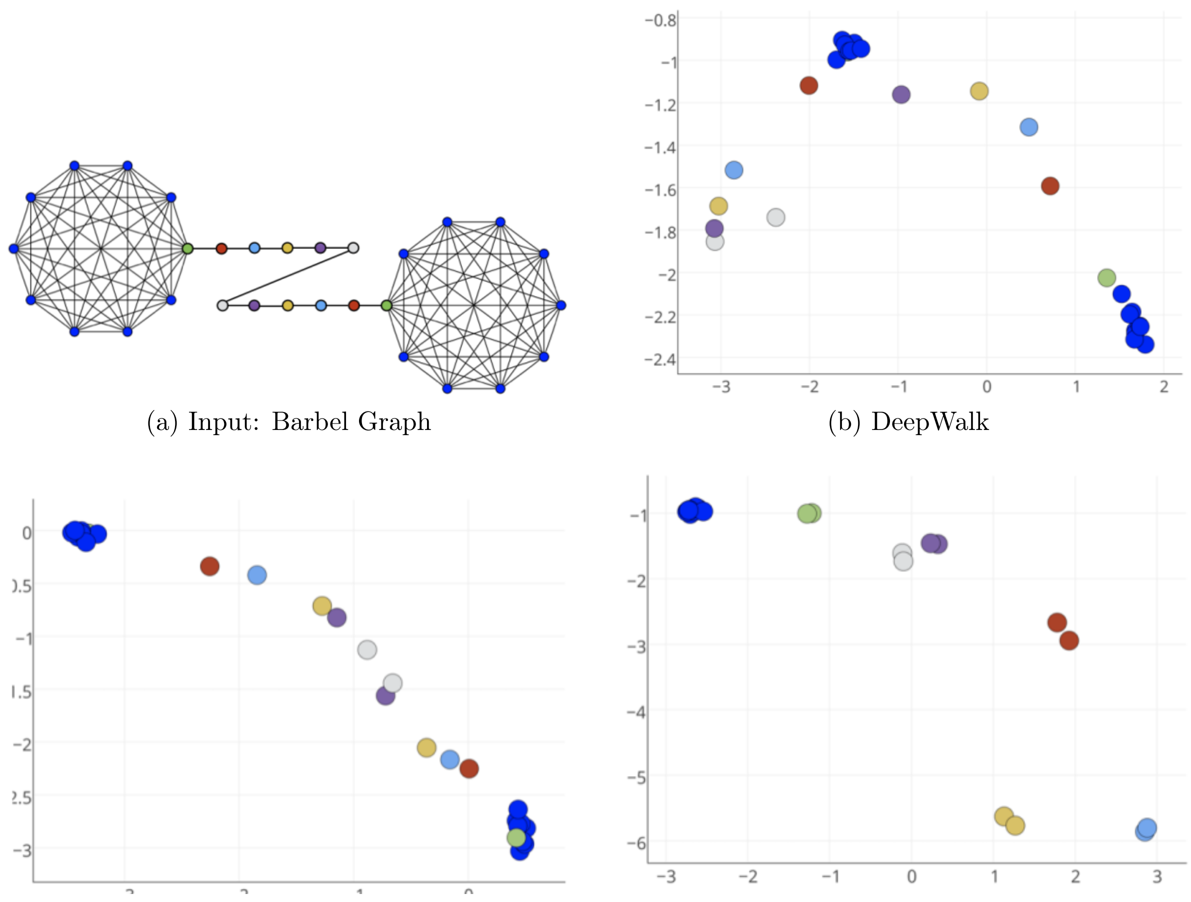
In addition to CNNs, Recurrent Neural network (RNN) [37] have also contributed to the development of Graph Neural Networks. Representation learning is another machine learning field that has influenced the development of GNNs. Their purpose is to learn latent, informative and low-dimensional vectors representations for graphs, nodes and edges, which can preserve the network structure, nodes features, labels and others informations.

DeepWalk [2] is regarded the first sucess methods based on representation learning. This unsupervised feature learning algorithm takes as graph as input and produces the latent representation as an output for every node as can see in the Figure 2.4. DeepWalk applies the language model SkipGram [38] on the sequence nodes generated by a stream of short random-walks. Similar approachs have been proposed as Node2vec [39] and Struct2vec [3]. Node2vec is a method that maps nodes based on neighborhood similarity, while Struct2vec maps nodes based on structural topology. The embeddings generated by these methods can be utilized for tasks such as node classification, edge prediction, and graph classification. Figure 2.5 depicts the variation in the latent representation across these methods within the same graph.



(a) Input: Karate Graph (b) Output: Latent Representation

Figure 2.4: DeepWalk applied on Karate Graph. Extracted from [2]



(a) Input: Barbel Graph (b) DeepWalk

(c) Node2vec (d) Struct2vec

Figure 2.5: DeepWalk, node2vec and struct2vec applied in Barbell graph $B(10, 10)$. Extracted from [3]

According to Hamilton et al. [40, 41], these methods also known as shallow embeddings, suffer from three significant limitations. The first limitation is that these methods are inherently transductive, which means they have difficulty in generalizing to unseen nodes during the training phase. For example, if a new node is added to the graph, the method needs to be reapplied to generate its corresponding latent representation. Second, is that they do not consider attribute information, which can be highly informative in many graph-based tasks. By neglecting the attribute data associated with nodes, shallow embedding methods may fail to capture the rich and meaningful information encoded in the attributes. Third, they do not share parameters between nodes in the encoder. The lack of parameter sharing leads in computational inefficiency, especially in large graphs, as the number of parameters grows linearly with the number of nodes. This inefficiency hinders scalability and can make the training and inference processes computationally expensive.

Graph Neural Networks (GNNs) were developed based on the premises of Convolutional Neural Networks (CNNs) and shallow embedding methods. Hamilton [40] defines GNNs as a framework for defining deep neural networks on graph data. The idea behind GNNs is to generate embedding vector representations for nodes, taking into account both the graph structure and any available feature information. Thus, GNN takes as a input graph $G = (V, E)$ along with a set of node features $X \in R^{d\times n}$ and uses this to generate node embeddings $h_u, \forall_u \in V$

Bronstein et al. [4] mention that the majority of research on Graph Neural Networks (GNNs) can be categorized into three main flavors of GNN layers: convolutional, attentional, and message passing, represented in Figure 2.6. Theses flavors represent different approaches to aggregating and propagate information across the graph. GNN architectures are characterized by being permutation equivariant functions $F(X, A)$, where X represents the node features vector and A is the adjacency matrix constructed by applying shared *permutation invariant functions* $\phi(x_u, X_{N_u})$ over the local neighbourhoods for every node $u \in V$. The function ϕ is also referred as diffusion, propagation, or message passing, and it constitutes the overall computation of F as a GNN layer. Permutation invariance, in all three flavors, is achieved by aggregating features from the neighborhood using a permutation-invariant function $\bigoplus$ and then updating the features nodes by some function ϕ. The non-parametric operation $\bigoplus$ can take various forms, such as sum, mean, or maximum, to aggregate features from the neighborhood. Additionally, the functions ϕ and ψ are learnable parameters.

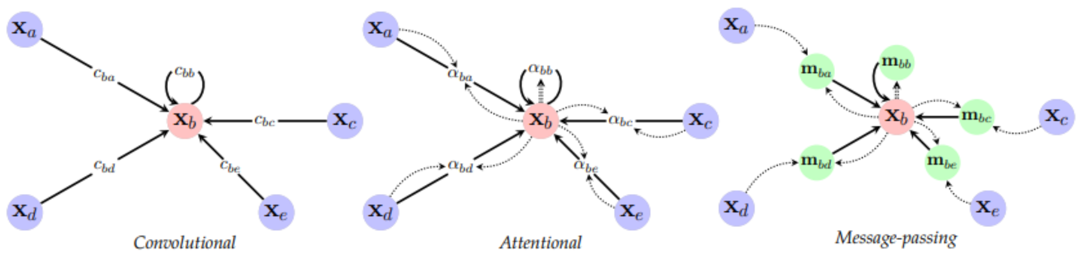


Figure 2.6: Convolutional and attention architectures aggregate the neighborhood representation vectors. In convolutional architecture, the node neighborhoods are aggregated according to a fixed weight parameter c. In attentional architecture, the aggregation of nodes is based on a learned parameter α. The message passing process, on the other hand, aggregates the information generated by the neighboring nodes. Extracted from [4]

In the **convolutional flavor** [42], the Convolutional Neural Network (CNN) operator is adapted to the graph domain. This flavor aggregate information from the neighborhood of each node by considering the node features of its neighbors. The aggregation is performed by taking the normalized sum of the node features of the neighboring nodes. This approach allows capture local patterns and dependencies within the graph structure. The convolutional operator is described as follow:

$$h_u = \phi\left(x_u, \bigoplus_{v \in N(u)} w_{uv}\psi(x_v)\right) \tag{2.5}$$

Note that w_{uv} represents the fixed weight that indicates the importance of the signal from node v to node u in the representation.

On the other hand, in the **attentional flavor**, as described below, an attention mechanism is incorporated, denoted by α, which assigns different importance to the signals coming from the node's neighborhood.

$$h_u = \phi\left(x_u, \bigoplus_{v \in N_u} \alpha(x_u, x_v)\psi(x_u, x_v)\right) \tag{2.6}$$

The last flavor, **message-passing**, as described below, computes the messages m_{vu} sent from every node v which belongs to the neighborhood of node u and then aggregates them using the permutation-invariant function $\bigoplus$.

$$h_u = \phi\left(x_u, \bigoplus_{v \in N(u)} \psi(x_u, x_v)\right) \tag{2.7}$$

2.4.2 Graph Neural Networks Architectures

Graph Convolution Network

Proposed by Kipf et al. [42], the Graph Convolutional Network (GCN) is a scalable approach for semi-supervised learning on graph-structured data, based on a variant of a convolutional neural network. The multi-layer Graph Convolutional Network (GCN) applies the following layer-wise propagation rule:

$$H^{(l+1)} = \sigma\left(\tilde{D}^{-\frac{1}{2}}\tilde{A}\tilde{D}^{-\frac{1}{2}}H^{(l)}W^{(l)}\right) \tag{2.8}$$

Note that:

$$H^{(0)} = X$$

Where $\tilde{A} = A + I$, where $I \in R^{n\times n}$ is the identity matrix with goal to add self-loops and the degree diagonal matrix with $\tilde{D}_{ii} = \sum_{u\in V}\tilde{A}_{ij}$. The matrix W^l of the l-th layer is a trainable parameter in GCN and σ is the activation function, such as the $ReLU(\cdot) = \max(0,\cdot)$ The last layer of embedding matrix $H^{(l)} \in R^{n\times F}$ contains in each row the corresponding node representation with dimension F.

The proposed GCN uses a two-layer architecture for semi-supervised node classification. In the first layer, operations are performed using the node features directly. The second layer utilizes the node embeddings obtained from the first layer. The model can be write by:

$$Z = f(X, A) = softmax\left(\tilde{A}ReLU\left(\tilde{A}XW^{(0)}\right)W^1\right) \tag{2.9}$$

In order to fit the parameters for the semi-supervised multi-class classification, the cross-entropy is evaluated over all the labeled examples.

$$L = -\sum_{l\in y_L}\sum_{f=1}^{F} Y_{lf}\ln Z_{lf} \tag{2.10}$$

where y_L is the set of node indices that have labels.

They used the batch gradient descent to trained the matrices W^0 and W^1. They utilize all the nodes during the training iteration phase.

GraphSAGE

Graph Neural Network (GNN) methods encounter scalability challenges as the complexity of the model increases with the number of nodes in the graph. In order to avoid this, Hamilton et al. [5] propose the GraphSAGE (Sample and AggreGatE), an inductive representation learning framework, in other words, it is not necessary to use all the nodes in the training phase to generate representations for unseen nodes.

The key idea behind GraphSAGE is to utilize a fixed neighborhood sampling strategy and an aggregator function. Instead of training a distinct embedding vector for each node, GraphSAGE trains a set of aggregator functions that learn to aggregate feature information from the local neighborhood of a node.

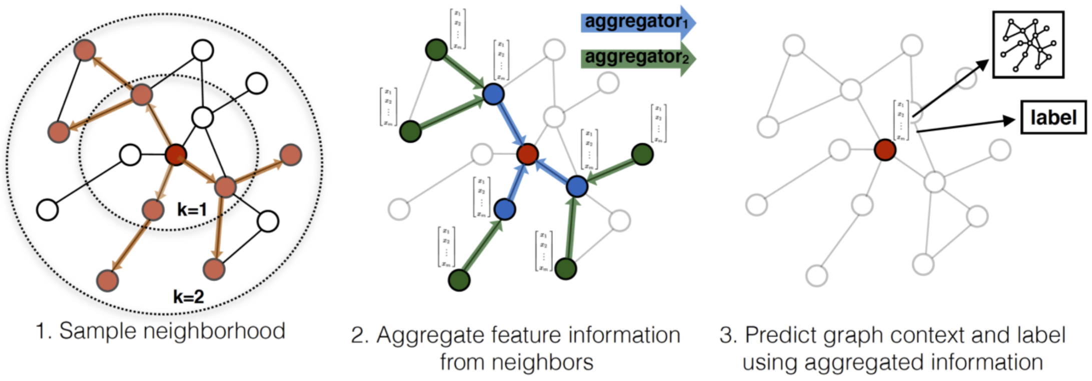

k=1
k=2
1. Sample neighborhood
aggregator₁
aggregator₂
2. Aggregate feature information from neighbors
label
3. Predict graph context and label using aggregated information

Figure 2.7: The GraphSAGE architecture follows several steps. First, a fixed neighborhood is randomly sampled for each node. Second, two aggregator functions are applied in different layers to aggregate the information from the neighborhood nodes. Third, a prediction function is applied to learn the model parameters and make predictions based on the aggregated node representations. Extracted from [5]

Figure 2.7 shows methodology of GraphSAGE, the first stage involves sampling the neighborhood of a node from two-hop away. Then, two aggregator functions are applied. The first aggregator function combines the features of the nodes that are 2-hops away (green nodes), while the second aggregator function combines the features of the immediate neighbors (blue nodes). Ideally, the aggregator function be invariant to permutation. Three kinds of aggregators functions are used: **mean, LSTM and pooling aggregators**.

The equation below represents an example of one layer of GraphSAGE with the MEAN aggregator function:

$$h_u^l = \sigma\left(W \cdot MEAN\left(CONCAT(h_u^{l-1}, h_v^{l-1}), \forall v \in N(u)\right)\right) \tag{2.11}$$

Note that h_v^l is the representation of a node v at layer l, W^l is the trainable weight matrix, $N(v)$ is the set of nodes sampled as neighbours of node v, and MEAN is the aggregation function that combines the feature of the neighbours. The CONCAT operation concatenates the aggregated features with the representation from the previous layer h_v^{l-1}, and σ denotes the activation function.

2.4.3 Community Detection with GNN

GNN has been successfully applied to node classification [43] and link prediction [44] in various scenarious [30, 31, 45]. Furthermore, GNNs have more recently been applied to community detection in attributed networks, where they have shown promising results [46].

GNNs are commonly employed in supervised or semi-supervised contexts, where the availability of fully or partially labeled data is used to train the model. GNNs are not initially designed specifically for the community detection task, which is an unsupervised problem. Many works apply GNN to community detection by formulating appropriate loss functions that encourage the model to learn representations that capture hidden information. These loss functions, such as reconstruction graph and attributes and contrastive loss, are examples that are widely applied to guide the training process. However, it is important to note that these loss functions do not directly optimize the model for community structure. Instead, they encourage the model to learn representations that capture relevant information for community detection, such as local graph structure or node similarities.

Many works employ GNNs as encoders to map nodes into an embedding space and then utilize the decoding process to reconstruct the graph and compare the original graph with the decoded graph. This reconstruction process serves as a training objective, where the model aims to minimize the discrepancy between the original graph and the decoded graph. By comparing them, the model can assess its ability to capture the important characteristics of the graph and uncover any hidden patterns or structures.

Recent works in community detection using graph neural networks have to propose loss functions that are more directly related to the community structure. These loss functions aim to optimize specific community-related objectives and enhance the detection of community structures. Two examples of such loss functions are entropy-based losses and modularity-based losses. Entropy-based losses encourage the model to produce more balanced and diverse community assignments by penalizing highly concentrated or imbalanced community distributions. Modularity-based losses, on the other hand, directly optimize the modularity measure, which quantifies the quality of community structures in a graph. By maximizing the modularity, the model is incentivized to find partitions of the graph that have a high density of edges within communities and a low density of edges between communities. By incorporating these loss functions into the training process, the models can explicitly focus on optimizing community-related objectives and improve the detection of community structures in an unsupervised manner.

Wang et al. [47] propose SGCN a GCN framework with a local label sampling. They argue that central nodes have a significant impact on the community structure of a graph. It is suggested that if we have knowledge of at least these central nodes, we can uncover or reveal the underlying community structure. To accomplish this, the approach first identifies K initial nodes with high structural centrality, which serve as the community centers. Then, the labels of these central nodes are propagated to the top t nearest nodes. This set of labeled nodes is used as the training set for the GCN model to learn their representations. Subsequently, the model assigns unlabeled nodes to their respective communities through GCN-based node classification. SGCN uses the shallow model proposed by Kipf [42].

Chu Wang et al. [6] proposed a Deep Neighbor-aware Embedded Node Clustering framework (DNENC) that utilizes the encoder-decoder GCN for community detection. The encoder aggregates information from the node neighborhood, and multiple layers of encoders are stacked to create a deep architecture for embedding learning. The decoder reconstructs the topological graph information using binary cross-entropy, as follows:

$$L_r = \sum_{i=1}^{n} loss(A_{i,j}, \tilde{A}_{i,j}) \tag{2.12}$$

Furthermore, due to the absence of label guidance, they designed a self-training module that guides the optimization procedure. After the encoder phase, they applies the Kullback-Leibler Divergence Clustering loss, which gradually optimizes the embedding for better representation.

$$L_c = KL(P||Q) = \sum_{i}\sum_{u} p_{iu} log\left(\frac{p_{iu}}{q_{iu}}\right) \tag{2.13}$$

Wheres q_{iu} measures the similarity between node embedding and the cluster center embedding. Initially, as the clusters are unknown, they applied the K-means algorithm to select the first cluster. After that, they iterative learn the clusters centers by using the Stochastic Gradient Descent. In the end, they jointly optimize the autoencoder embedding and clustering learning with their respective objective function:

$$L = L_r + \gamma L_c \tag{2.14}$$

Where γ is a coefficient which controls the trade-off learning between these two loss functions.

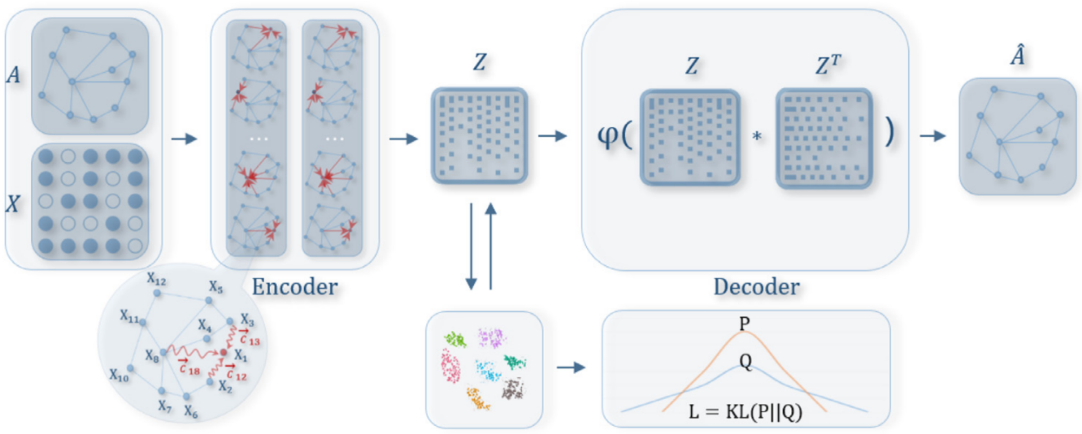


Figure 2.8: The schematic of the Deep Neighbor-aware Embedded Node Clustering (DNENC) framework consists of several components. First, an embedding matrix Z is generated by a GNN-autoencoder, which reconstructs the graph structure and captures the latent representations of the nodes. Next, the embedding matrix Z is manipulated using a self-training clustering module, which optimized together with the autoencoder during training and performs clustering based on the learned representations. Extracted from [6]

Sambaran and Vishal [7] combined the principle of self-expressiveness with the framework of self-supervised learning to propose SEComm (Self-Expressive Community detection in graph).

The key idea is to generate two graphs, denoted as G_1 and G_2, by applying a corrupted function that randomly removes a small portion of edges. These corrupted graphs are then used for contrastive learning, considering both graph topology and node features. The GCN encoder, based on the work of Kipf et al. [42], is trained with the following objective function:

$$L_{SS} = \sum_{i \in V} \left(-\frac{\cos\left(Z_{1i}, Z_{2i}\right)}{\tau} + \log\left(\sum_{j \in V_{-i}} e^{\frac{\cos\left(Z_{1i}, Z_{1,j}\right)}{\tau}} + e^{\frac{\cos(Z_{1i}, Z_{2j})}{\tau}} \right) \right) \tag{2.15}$$

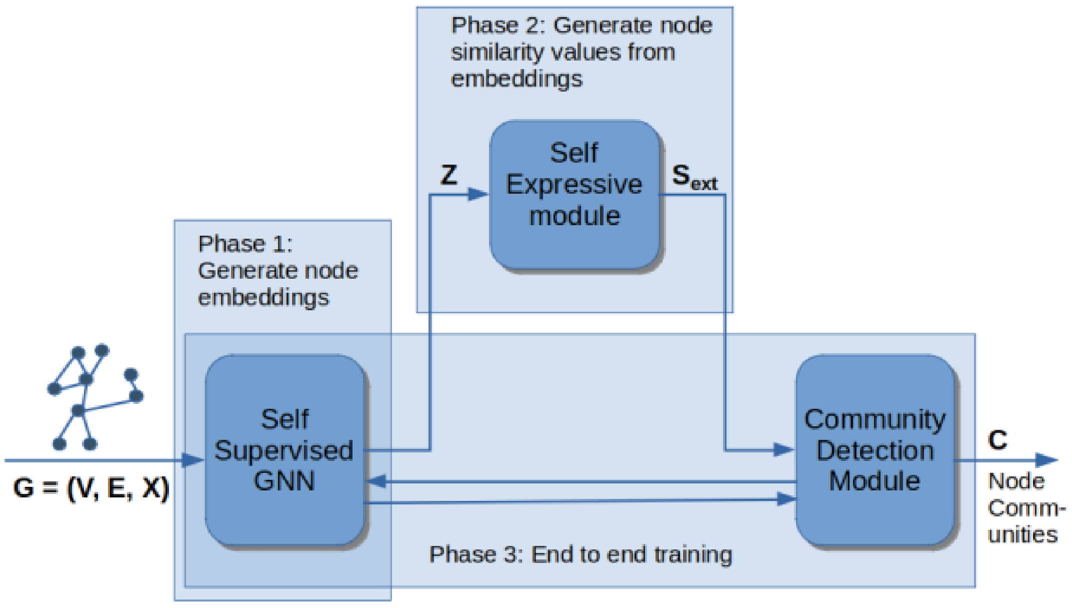

Phase 2: Generate node similarity values from embeddings
Self Expressive module
Z
S_ext
Phase 1: Generate node embeddings
Self Supervised GNN
G = (V, E, X)
Community Detection Module
C
Node Comm-unities
Phase 3: End to end training

Figure 2.9: The schematic of the SEComm (Self-supervised Embedded Community Detection) framework involves several steps. First, a self-supervised Graph Neural Network (GNN) is trained using two corrupted versions of the original graph, applying the self-expressive principle to generate the node embeddings Z. Next, a community detection step is performed using a spectral clustering loss function. Extracted from [7]

With the node embeddings obtained, a fully-connected multi-layer perceptron (MLP) is used to map each node embedding to its corresponding soft community membership. This mapping is performed as follows:

$$C_i = softmax\left(MLP(Z_i)\right) \tag{2.16}$$

In order to capture meaningful signals from the clustering assignment, the proposed framework incorporates a community detection objective function.

$$L_{com} = \sum_{(i,j)\in S_ext} \left(C_i^T C_j - S_{ij}\right)^2 + \lambda_2 || \frac{C^T C}{||C^T C||_F} - \frac{I_K}{\sqrt{K}} ||_F^2 \tag{2.17}$$

The total loss used to train SEComm is calculated as a weighted sum of the self-supervised loss and the community detection loss.

$$L_{total} = \alpha L_{SS} + L_{Com} \tag{2.18}$$

Self-supervised learning has been frequently employed in frameworks that aim to tackle the task of community detection. Zhang et al. [29] proposes CommDGI, a framework which uses self-supervised learning to encode nodes. They employed three distinct objective functions in order to capture the community information. First, a contrastive method is employed to capture structural similarities. This involves selecting both negative and positive samples and training the model using a binary cross-entropy loss to learn hidden structures. Second, a differentiable K-means algorithm is applied to refine the clustering results. This step aims to optimize the assignment of nodes to different communities based on their learned representations. Third, a modularity objective is utilized to capture more edge information and graph-level partition information. This objective function helps to further improve the community detection by maximizing the modularity score, which quantifies the quality of the community structure in the graph. The model was adjusted by the weighted sum of the three losses.

Muller et al.[12] introduce Deep Modularity Networks (DMoN), an unsupervised framework inspired by the modularity measure. They make two modifications to the classic GCN architecture proposed by Kipf[42]. First, they remove the self-loop and introduce a trainable skip connection W_{skip}. Second, they use the Scaled Exponential Linear Unit (SeLU) activation function instead of Rectified Linear Unit (ReLU). SeLU is a self-normalizing activation function that helps mitigate the vanishing gradient problem and improves the model's performance. To train the model, they propose a modularity loss function with collapse regularization. This loss function aimed to maximize the modularity score, which measures the quality of the community structure in the graph. The collapse regularization term was introduced to prevent trivial solutions and enhance the generalization capability of the model.

Lian et al. [28] propose a framework called Multilayer Graph Contrastive Clustering Network (MGCCN) for community detection. The framework consists of three modules. The first module incorporates an attention mechanism to enhance the capturing of the relevance between nodes and their neighbors. This attention mechanism allows the model to focus on the most informative neighbors when learning the node embeddings. The second module introduces a contrastive fusion strategy to improve the clustering performance. This strategy leverages contrastive learning to encourage similar nodes to have similar embeddings, while dissimilar nodes have different embeddings. By doing so, the model learns to group similar nodes together in the clustering process. The third module is a self-supervised learning component that iteratively adjusts the node embeddings and the clusters. This iterative process refines the embeddings and cluster assignments, leading to improved clustering results. Overall, the MGCCN framework combines attention mechanisms, contrastive fusion, and self-supervised learning to enhance the clustering performance in community detection tasks.

Note that the mentioned frameworks adopt self-supervised learning as a mechanism to guide the optimization process and improve the quality of the detected clusters at each iteration. By leveraging self-supervised learning techniques, these frameworks can learn from the intrinsic structure of the data and make use of unsupervised signals to refine the community assignments. This allows for the discovery of more accurate and meaningful community structures without the need for explicit labels or supervision.

Both of these algorithms were tested on the three benchmark datasets: Cora, Citeseer, and PubMed. However, as mentioned in Section 2.2, this limited analysis makes it difficult to determine which framework is better or whether they suffer from overfitting.

Our work introduces a novel approach to learning node representations for community detection in attributed graphs. Unlike theses methods, we do not rely on specific loss functions. Instead, we employ a dynamic graph and a self-learning module that iterative optimize the node embeddings. This approach allows to refine the dynamic graph based on the evolving node representations, enhancing community detection performance. Through extensive experiments on various datasets, including real-world attributed graphs, we demonstrate the effectiveness and robustness of our approach. The results show significant improvements in community detection performance compared to existing methods, highlighting the potential of our approach in uncovering meaningful community structures in attributed graphs.

Chapter 3

Proposed Framework: Dynamic Context Self-Learning-GNN

This chapter presents and describes the framework Dynamic Context Self-Learning Graph Neural Network (DCSL-GNN), a fully unsupervised framework for clustering attributed networks. DCSL-GNN comprises three interconnected modules: Dynamic Context, Embedding, and Clustering. The Dynamic Context module first constructs a new network by performing multiple biased random walks on the original graph. Second, the Embedding module utilizes GNN to generate latent representations of nodes considering both the new network structure and fixed node attributes. Finally, the Clustering module applies a classic data point algorithm in the nodesohe Embedding module. Note that clustering task is intrinsically unsupervised, i.e., there is no prior knowledge concerning the node cluster labels. Additionally, GNNs are sometimes trained using supervision. To address this challenge, we employ a novel learning paradigm known as *self-learning* to train the model. Thus, the modules Context Generator, Embedding and Clustering are jointly learned in an end-to-end manner using several iterations.

3.1 Introduction

Given an attributed graph undirected $G = (V, E, X)$, where $V = \{v_1, v_2, \dots, v_n\}$ is a set of n nodes and E is a set of edges with $e_{ij} = (v_i, v_j) \in E$ if an edge exists between nodes v_i and v_j, and $X = \{x_1, x_2, \dots, x_n\} \in \mathbb{R}^{n \times F}$ is the attribute matrix, where each node v_i is associated to an attribute vector x_i with dimension F. The problem under consideration is partition the nodes into disjoint sets accordingly to node topological structure and their attributes. In order to solve this problem, this thesis proposes a novel model to learn a function $f : V \to C$, where $C = \{C_1, C_2, \dots, C_K\}$ is the set of communities (or clusters), which maps each node into a non-overlapping community C_i. This model learns without any supervision and the only information available is the attributed graph G and number of communities, K.

Important notation used in this work is described in Table 3.1.

Notations	Definition
$G = (V, E, X)$	Attributed graph
$G_c = (V_c, E_c, X)$	Context graph
$X \in \mathbb{R}^{n \times F}$	Node feature matrix
$H(\cdot)$	Embedding for node v
W	Learning matrix
$C^l \in \mathbb{R}^K$	Community sets at l-th round
$\|\|X, Y\|\|$	Euclidean distance between X and Y
$L(\theta)$	Triplet loss function
K	Number of communities
$N(v)$	Neighborhood of node v
$s(v)$	Silhouette value of node v
n_e	Number of epochs
n_u	Mini-batch size

Table 3.1: Notation.

Recently, several works have been proposed frameworks that uses GNNs for clustering attributed networks [6, 7, 12, 27, 47–49]. An illustrative example of an architectural framework can be observed in Figure 3.1. In this approach, the generation of embeddings and the clustering task are separated. Specifically, nodes are initially mapped into Euclidean space, and subsequently, a clustering algorithm is applied to these latent representations. It is important to note that the learning model employed in generating the latent representations does not benefit from the clustering phase. In other words, the cluster information is not used for node mapping. This decoupling may potentially result in sub-optimal representations for the clustering task.

The framework proposed by this work has an architecture represented by Figure 3.2. In this architecture, the graph used as input for generating embeddings is not original graph, but a graph constructed and referred to as the Context Graph (G_c). The context of a node is determined by the set of most frequently visited vertices, according to multiple biased random walks on the original graph. This technique is commonly employed in other algorithms like PinSage [34] in order to determine the importance of neighboring nodes.

In the Context graph, nodes that are not neighbour in G can be selected to the neighbourhood in G_c. For example, the node v in Original Graph, after multiples random walks, nodes y, x, z were the most visited nodes, and they were included in the neighborhood of node v in G_c. Note that in the embedding phase, the latent vectors exhibit characteristics of both classes, but certain characteristics may have a

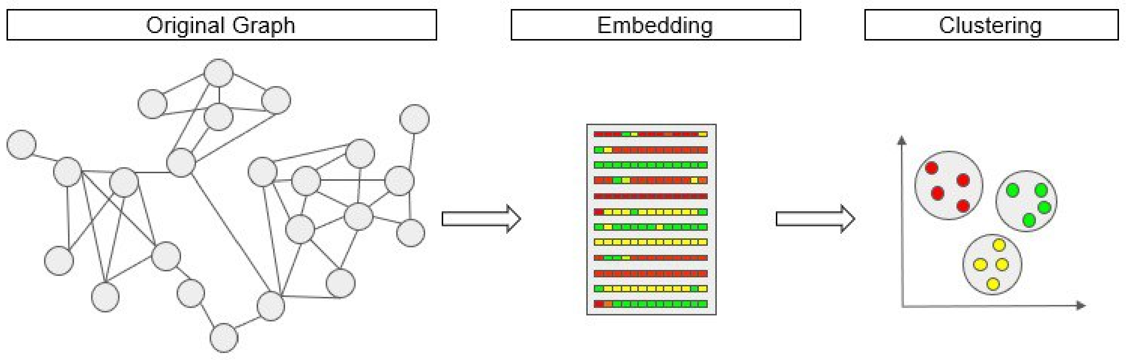


Figure 3.1: This framework model is commonly used and is characterized by the decoupling of the embedding generation and clustering phases. In this approach, the two phases are independent of each other and not directly benefit from each other's results.

stronger influence than others. For example, in the case of the first node, it possesses more characteristics that align with the red class compared to the other classes. This implies that its representation in the latent space is closer to the red class, making it easier for the clustering algorithm to identify the corresponding cluster. Finally, the latent space representations and soft-assigned clusters obtained in the current round are reused in the subsequent round.

DCSL-GNN for clustering operates in rounds, with its primary contribution being the self-learning component and the iterative generation of context throughout the rounds. At each round of the algorithm, the context of nodes is readjusted based on the results of the previous round embedding and clustering phases. The contexts graph allow for nodes that are very representative vertices of a cluster to become neighbors with others nodes in that cluster, even if they are not neighbors in the original graph. By considering a broader notion of neighboring vertices, the framework aims to capture a more comprehensive representation of the graph structure and node attributes, thus improving the quality of the generated embeddings for the task of clustering.

The core idea is to consider vertices that are up to k hops away candidates for the the context of a given vertex, as long as these vertices are representative of a cluster. This approach enables information from the center of the cluster to be propagated more directly to the vertices, of the cluster as opposed to traveling through intermediate vertices.

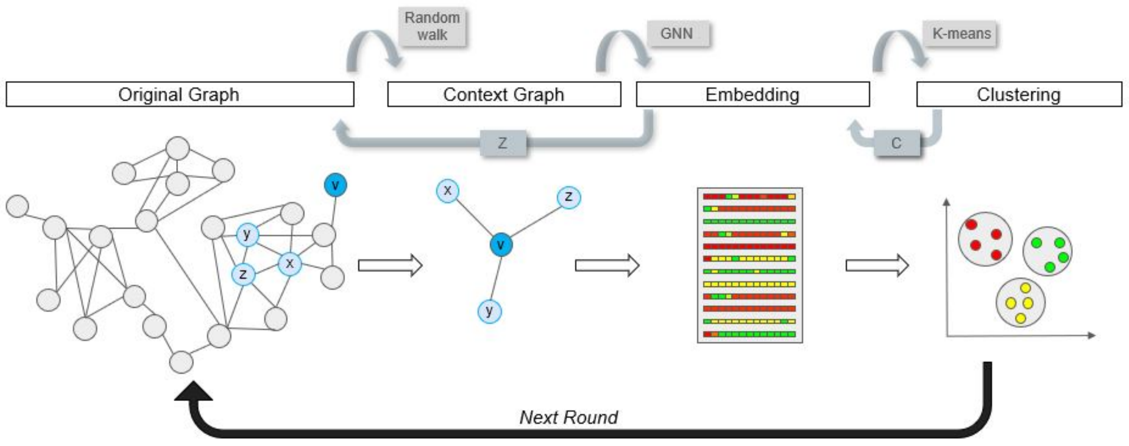


Figure 3.2: Schematic representation of DCSL-GNN framework. Note that the clustering result is used in the next round to build a new contex graph. Thus, embeddings influence clustering, and clustering influences embeddings.

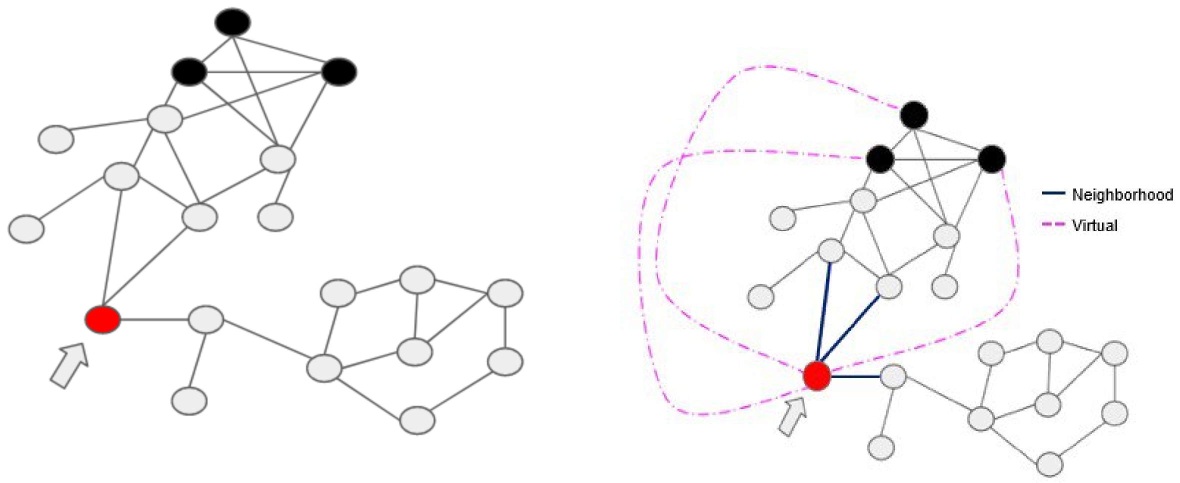
3.2 Context Generator

GNNs aim to learn node embeddings by integrating attributes with graph structure. In GNN, a layer can be viewed as a message passing between nodes, where each node updates its representation (embedding) by aggregating the messages from its direct neighbors with their respective representations.

In the context of the graph clustering problem, nodes most representative of a cluster could be more than K hop distance. This imposes a challenge for GNNs to effectively propagate information from these nodes to others. The reason behind this challenge arises from the fact that, with the objective of allowing a node to receive information from nodes within a radius of K, the depth of GNN layer needs to be at least K. Without this depth, the information propagation is constrained, resulting in a phenomenon known as under-reaching [50]. Nevertheless, unlike conventional neural network algorithms, when additional layers are stacked in GNNs, the nodes representation tends to become increasingly indistinguishable. Consequently, this leads to a significant degradation in prediction accuracy and overall performance, known as over-smoothing [42, 51, 52]. For that reason, most GNN models have very few layers, such as 2 or 3, and often deploy mechanism to avoid over-smoothing.

Due to limitations of the node neighborhood, which could be not representative to effectively propagate cluster signals and in order to mitigate under-reaching and over-smoothing issues, we propose generating a new graph that feeds the GNN. By modifying the node's neighborhoods with a new set of edges, we aim to improve the cluster information flow and ensure that relevant cluster signals are properly propagated. The goal is to create a graph structure that better represents the relationships between nodes belonging to the same cluster. Hence, we designed a flexible neighborhood sampling strategy biased of the original graph by the similarity of the representation of the nodes generated in the previous round.

Additionally, to facilitate the direct propagation of cluster information to individual and representative nodes, we introduced the concept of "virtual edges". The virtual edges connect a node to a fixed number of nodes, denoted as T, that are closest to the cluster center of mass in the embedding space as illustrated in, Figure 3.3. Thus, with a certain probability, the neighborhood of a node can be sampled from these nodes. This approach enables the aggregation of both information from the cluster center through virtual edges and the local structure of a node from the original edges of the graph. Combining these two sources of information, a more comprehensive graph representation is created with respect to cluster signals and local structure. The resulting graph is referred to as Context Graph (G_c).



(a) Original Graph (b) Random walk perspective

Figure 3.3: In each step of the random walk, a decision is made to either follow a virtual edge towards the cluster center nodes or explore the neighborhood of the current node.

As previously mentioned, the DCSL-GNN framework operates in a self-learning manner. This mechanism allows the framework to learn from its own predictions and iteratively update the model parameters to better capture the cluster structure. In each round of training, the node embeddings and soft-label assignments from the previous round are reused to guide the learning process of the current round. This iterative process gradually improves the clustering accuracy (as shown in empirical results).

DCSL-GNN employs random walk as a mechanism for generate the sequence of nodes that determine the context of a given node, as adopted by several works [3, 34, 39]. We propose a random walk mechanism that captures the similarities between nodes in terms of cluster membership. This means that a node will move to another node if they share a higher degree of similarity in terms of their cluster assignments. As we introduced the concept of virtual edges, during each step of the random walk, a decision is made to either traverse a real edge in the original graph or select a virtual edge leading to a node in the cluster center. This choice is determined by a Bernoulli process, where an original edge is taken with a probability of p, and a virtual edge is chosen with probability $(1 - p)$. If the random walks selects a virtual edge, the node in the center of the cluster is chosen by a uniform probability. Otherwise, when an original edge is chosen, the random walk explores the neighborhood based on the proximity in the embedding space. Thus, the random walk is biased and edge weights are inversely proportional to distances between their representations.

To guide the random walk in the original graph, an edge weight function denoted by $w(\cdot)$ utilizes the node representations generated in the previous round to bias the walk. The value of $w(\cdot)$ increases as the Euclidean distance between nodes representations becomes smaller. This behaviour directs the random walk towards nodes that are more likely to lead to nodes with similar embeddings, ensuring that the exploration of the random walk focuses on graph nodes that are closer together in the embedding space.

The Equation 3.1 presents the edge weight function. It is important to note that the value $\max_{z \in N(v)}(||H(v), H(z)||)$ represents the longest distance from node v to its neighbors in the embedding space. The value of edge weight tends to be higher for nodes closer to v, approaching $\max_{z \in N(v)}(||H(v), H(z)||)$, and tends to be lower for nodes that are farther away, approaching 0 (note that it is zero for the farthest node). In a nutshell, the weight function $w(\cdot)$ corresponds to the Euclidean distance relative to node v.

$$w_H(v, u) = \max_{z \in N(v)}(||H(v, z)||) - ||(H(v), H(u))|| \tag{3.1}$$

$H(\cdot)$ is is the latent node representation for the nodes in euclidean space.

Given a node u, we simulate multiples random walks of length L biased by $w(\cdot)$ in order to select nodes to compose the context for node u. Let c_i denote the i-th node in the walk, starting with $c_0 = u$. Nodes c_i are generated by the following distribution:

$$P(c_i = w | c_{i-1} = v) = \begin{cases} w_H(v, w)/Z_v, & w \in N(v), \text{w.p. } p \\ 1/T, & w \in CC(u), \text{w.p. } 1 - p \end{cases} \tag{3.2}$$

where $CC(u)$ is the set of nodes in the cluster center of node u, and Z_v is the normalization factor for vertex v given by:

$$Z_v = \sum_{i \in N(v)} w_H(v, i) \tag{3.3}$$

To determine the neighborhood of node u in the Context Graph (G_c), denoted as $N_c(u)$, we adopted Importance-based neighborhoods, as proposed in the PinSage algorithm [34]. This method defines the neighborhood of a node u as the T nodes that exert the most influence on node u. As the embeddings are influenced by both the network structure and attributes, using them to bias the random walks generates a sequence of nodes that are strongly correlated with the clusters they belong to, due to the homophily phenomenon. In particular, the most visited nodes will be the neighbors of node u. Thus, the visits counts of the random walks are used to select the most important nodes. Since each source node $u \in V_G$ is associated with an independent non-identically visit count distribution, we employ a threshold value to determine if a visited node will be included in the new neighborhood. This threshold value is defined as the k-th percentile of the distribution, determined by a parameter. Consequently, the neighborhood of node u, denoted as $N_c(u)$, in the Context Graph is comprised of nodes that have been visited at least the value of the k-th percentile.

Note that constructing node neighborhoods based on importance has several advantages. First, the selection of the most important nodes is based by visits counts, it enables nodes with high visit counts being direct neighbors in the Context Graph, even when the node is many hops away, consequently, it reduces the noise when passing messages between them. Second, a few layers stacked in GNN in the original graph could already incur unwelcome messages between clusters, however this is less likely to occurs in the context graph, because the neighborhood contains only similar nodes that are likely to belongs to the same clusters. Third, as only the most important nodes are chosen, it allows to reduce the neighbourhood size without loss of generality and helps to decrease the algorithm memory's consumption.

3.3 Learning Embeddings

The encoder proposed in [42] was adopted in the Embedding Module. Given a graph G with a adjacency matrix A a stack layer of the model is defined as follow:

$$H^{(l+1)} = \sigma\left(\tilde{D}^{-\frac{1}{2}}\tilde{A}\tilde{D}^{-\frac{1}{2}}H^{(l)}W^{(l)}\right) \tag{3.4}$$

Where $\tilde{A} = A + I$, where $I \in \mathbb{R}^{n\times n}$ is the identity matrix with goal to add self-loops and the diagonal degree matrix with $\tilde{D}_{ii} = \sum_{j\in V} \tilde{A}_{ij}$. The matrix W^l of the l-th layer is a trainable parameter in GCN and σ is the activation function, we use *ReLu* in ours experiments. In the last layer of embedding matrix $H^l \in R^{n\times F}$ is the representation matrix and output from the previous layer and contains in each row the corresponding node representation with dimension F.

The operator can be defined in the perspective of a node as follow:

$$H_i^{(l+1)} = W^{(l)} \sum_{j\in N(i)} \frac{e_{j,i}}{\sqrt{\tilde{d}_j\tilde{d}_i}} H_j^{(l)} \tag{3.5}$$

where $e_{j,i}$ denotes the weight edge value from source node j to target node i and $\tilde{d}_j$ denotes the node j degree.

Initially, all edges in G_c have a weight value of 1, i.e, all messages from the neighborhood carry equal importance during the aggregation step. However, due the stochastic nature of context graph, uncertain edges can be part of the neighborhood, which can introduce noise and potentially mix their latent representations. Furthermore, nodes that exhibit high similarity in their attributes should contribute with greater importance during the aggregation step. In this way, an edge $e(u, v)$ is considered uncertain when the attributes from nodes u and v have no correlation with each other and at least one of them is highly correlated with the attributes of the cluster which it belongs.

To measure the cohesion of attributes within a cluster, the Silhouette coefficient algorithm [53] is used. This coefficient measures the similarity of a data point to its own cluster compared to other clusters. It considers both the distance between the data point and other points within its own cluster (cohesion) and the distance between the data point and points in other clusters (separation).

Assuming the node embeddings have been clustered into C clusters using any clustering algorithm, the silhouette coefficient for each data point i can be defined as follows:

$$s(i) = \frac{b(i) - a(i)}{\max(a(i), b(i))} \tag{3.6}$$

Where $a(i)$ is the average intra-cluster distance, i.e the average distance between each point within the cluster of node i and $b(i)$ is the average inter-cluster distance, which is the average distance between the node i and all points in other clusters, defined as follows:

$$a(i) = \frac{1}{|C_{z_i}| - 1} \sum_{j \in C_{z_i}, i \neq j} d(i,j) \tag{3.7}$$

$$b(i) = \min_{y \neq z_i} \frac{1}{C_y} \sum_{j \in C_y} d(i,j) \tag{3.8}$$

Where $d(i,j)$ represents the Euclidean distance between the data points i and j in the Euclidean space and C_{z_i} is the cluster of node i and z_i is the cluster number of node i.

For each node u in a cluster $-1 \leq s(u) \leq 1$. A silhouette value closer to 1 indicates that the data is appropriately clustered, a value closer by -1 indicates poor clustering, and a value closer to 0 suggests the value is on the border between multiples clusters.

The dynamic nature of context graph enables the utilization of the previous clustering results to guide the current optmization process. Thus, DCSL-GNN evaluate the G_c edges weigths in current process with the cluster obtained by the previous round. However, this optmization process might take a considerable amount of rounds to discover consistent clusters. Thus, the attribute $X[i]$ of a node i is considered properly clustered only if $s(i) \geq \beta$, where is β is a threshold value. If one of the nodes from an edge satisfies this condition, the edge weight value will depend on the inverse Euclidean distance between the attribute values of the nodes, otherwise the weight is considered 1.

For this, we propose a function F describe as follow:

$$e(j,i) = \begin{cases} \min\left(\frac{1}{\|X[i], X[j]\|}, \alpha\right), & \text{if } s(i) \geq \beta \text{ or } s(j) \geq \beta \\ 1 \end{cases} \tag{3.9}$$

An edge $e(i,j)$ with nodes attributes $X[i]$ and $X[j]$ assumes values between $[0, \alpha]$. It is important to note that α is a parameter that controls the upper bound of the weight, in order to prevent the edge weight from approaching infinity and dominating the integration. Therefore, when $e(i,j) < 1$, the edge is penalized as it reduces the importance between the nodes. Conversely, when $e(i,j) > 1$, it increases the influence in the messages between them.

3.4 Clustering Module

From the node embeddings provided by Embedding Module, a data clustering algorithm is applied. Several algorithm could be employed in this module, such as spectral clustering [54], hierarchical clustering [55] or K-means [56]. We adopted the K-means in DCSL-GNN, that is a popular centroid-based algorithm. This algorithm was proposed over 50 years ago and is perhaps the most ubiquitous approach to tackle data clustering.

K-means assigns a set of data points $X = \{x_1, \ldots, x_n\}$ into K disjoints sets $C = \{C_1, \ldots, C_k\}$ by iteratively fitting the data point to the nearest centroid and updates the centroid positions until convergence minimizing the objective function.

The objective function is defined as follows:

$$L = \sum_{n=1}^{N} \sum_{k=1}^{K} r_{nk} ||x_n - \mu_k||^2 \tag{3.10}$$

This optimization is achieved through the estimation of the parameters μ_k and r_{nk} that minimizes the objective function. These parameters are, respectively, the mean data points that belongs to the cluster k and an indicator variable that indicates when node n belongs to cluster k, as described, respectively, in Equations 3.11 and 3.12. Note that if one of the parameters is known, the other can be inferred. For this, an iterative method is applied in order to optimize these parameters.

$$\mu_k = \frac{\sum_n r_{nk} x_n}{\sum_n r_{nk}}, \text{for } k = 1, \ldots, K \tag{3.11}$$

$$r_{nk} = \begin{cases} 1 \text{ if } k = \arg\min_j ||x_n - \mu_j||^2, \text{for } k = 1, \ldots, K \\ 0 \text{ ow.} \end{cases} \tag{3.12}$$

To find the proper values r_{nk} and μ_k an iterative method is applied that involves optimizing these variables successively. In the first step, starting from initial values for μ_k, K-means minimize the equation 3.10 with respect to r_{nk}. In the second step, K-means minimize the equation 3.10 with respect to μ_k with the optimized values of the variable r_{nk} obtained in the previous step. These two steps are performed until convergence. The alternating optimization of r_{nk} and μ_k are the E (Expectation) and M (Maximization) steps of the Expectation Maximization (EM) algorithm [57]. Once the algorithm converges, r_{nk} is used to determine the sets C_k. In particular, $C_k = \{i | r_{ik} = 1\}$ for $k = 1, \ldots, K$.

3.5 Trainning

An important contribution of this work is the inclusion of a self-learning component in the proposed framework. In the training stage the process involves passing through each module, Dynamic Context, Embedding, and Clustering, in iterative rounds, as illustrated in Figure 3.2. For every round, DCSL-GNN train the GNN model using a different context graph that is constructed based on the original graph with nodes embeddings founded in the last round. In a round, the GNN in the Embedding module is trained for a fixed number of epochs to optimize and learn the weights matrices and bias for each layer. For each epoch, Context Graph (G_c) is considered, as opposed to a sample the entire network. However, we evaluate the loss function and update the parameters by $\frac{n}{n_b}$ times using mini-batches of size n_b which are uniformly sampled without replacement from the graph nodes. Despite using all nodes and edges in message passing, only the nodes in the batch are used to evaluate the loss function at each mini-batch. This approach allows us to make efficient use of the entire network while still updating the model parameters in a batch-wise manner, an approach known to be lead to better convergence.

Algorithm 1 illustrates the DCSL-GNN framework. In the very first round, before the clusters are formed, each node is considered to be in its own individual cluster (cluster C^0). Thus, during the first random walk procedure, each node in the neighborhood has an equal probability of being visited, and the walk can return to the given node (as the node is the cluster center) with probability $(1-p)$.

Algorithm 1 DCSL-GCN: model training using mini-batches for the evaluation function

```
Input
    G   Original Graph
    K   Number of Clusters
Output
    C   Clusters sets
    H   Nodes Embeddings
for $r = 1, \dots, n_r$ do
    $G_c \leftarrow$ GenerateContext($G, H^{r-1}, C^{r-1}$)
    for $e = 1, \dots, n_e$ do
        $H^r = GenerateEmbeddings(G_c, W)$
        for $i = 1, \dots, \frac{n}{n_b}$ do
            S $\leftarrow$ GenerateSeeds($V_c, n_u$)
            L $\leftarrow$ ComputeLoss($S, H^r$)
            UpdateWeights(W,L)
        end for
    end for
    $C^r \leftarrow$ GenerateClusters($H^r$)
end for
```

For trainning the GNN model, a loss function based on triplet loss was adopted, as shown in Equation 3.13. In this function a sample data point (known as anchor) is compared with the matching input (positive sample) and non-matching input (negative sample). Nodes that belong to the same cluster as the anchor node are considered positive samples, denoted as x_n^+. Nodes that do not belong to the same cluster are considered negative samples, denoted as x_n^-. This function aims to minimize the distance between an anchor and a positive sample; and maximizes the distance between the anchor and a negative sample. The hyper-parameter Δ enforces a margin distance between positive and negative pairs.

$$L_{triplet}\left(\Theta\right) = \frac{1}{N}\sum_{n=1}^{N} \max\left(||x_n - x_n^+||^2 - ||x_n - x_n^-||^2 + \Delta, 0\right) \tag{3.13}$$

This loss function was adopted because it aims to minimize the distances between similar data points (positive pairs) and increase the distances between dissimilar data points (negative pairs). This behavior is aligned with the procedure to construct the Context Graph, as the random walks are biased by the Euclidean distance in the latent space. Therefore, in every round, similar nodes in embedding space becomes closer and distinct nodes becomes farther apart away. This process contributes to create a better context in the subsequent round.

Finally, the relatively simple GCN architecture proposed by [42] was adopted. It a simple model with only two layers and few trainable parameters (only the weights and bias matrices). A GCN model with only two layers is considered, and thus, we need to learn two matrices, $W^{(0)}$ and $W^{(1)}$ as seen in the Equation 3.4. In each round of training, the parameters of the GCN for round r are initialized with the parameter values obtained from the previous round $r-1$. We can utilize learning algorithms such as Stochastic Gradient Descent (SGD) [58] or the Adam Optimizer [59]. The computational cost of a simple model is reduced and the framework becomes relevant in scenarios where the graph has a large number of vertices and edges. Moreover, this procedure must be efficient since a model will be trained in every round of the proposed framework.

Chapter 4

Evaluation

This chapter presents the methodology used to evaluate the performance of the proposed framework, DCSL-GNN, in various scenarios. The evaluation is conducted using synthetic attributed networks generated by the method described in Section 2.3 as well as benchmark datasets used by the related frameworks described in Section 2.4. The objective is to demonstrate the capability of the framework in accurately clustering nodes based on their attributes and network structure.

4.1 Syntethic Graphs

With the objective of understanding the limitations of our proposed framework, we generated multiple graphs with different community structures and attribute distributions. These graphs were designed to simulate various scenarios and test the performance of the DCSL framework. In each scenario, we carefully varied the parameters related to community structure, such as the interconnectivity between them. We also considered different attribute distributions to mimic real-world datasets with diverse characteristics.

4.1.1 Metric

Evaluating the cluster quality outputs from any method can be challenging in unsupervised learning, as there is typically no ground truth or labeled data available to directly compare the results with. However, there are several metrics and techniques that can be used to assess the cluster quality. In our analysis, since we have access to the ground truth labels of the datasets, we have chosen to utilize the Normalized Mutual Information (NMI) [60] as an evaluation metric.

NMI measures the mutual information between the predicted cluster assignments and the true class labels, normalized by the entropy of both assignments. The value ranges from 0 to 1, where a higher value indicates a perfect match between the predicted and the ground truth labels, and a value close to 0 indicates a lack of agreement. Inspired by Shannon Entropy [61], the Mutual Information, represented by $I(\cdot)$, between two discrete variables X and Y is defined as follows:

$$I(X;Y) = \sum_{y \in Y} \sum_{x \in X} P(x,y) \log \left(\frac{p(x|y)}{p(x)} \right) = H(X) - H(X|Y) \tag{4.1}$$

with:

$$H(X) = E[-\log P(X)]$$

$$H(X|Y) = \sum_{y \in Y} P(y) H(X|Y=y)$$

where $H(X)$ is the entropy of the random variable X and $H(X|Y)$ is the conditional entropy.

In order to normalized the Mutual Information to values between 0 and 1, the measure can be defined as follow:

$$NMI(X,Y) = \frac{2 * I(X,Y)}{H(X) + H(Y)} \tag{4.2}$$

4.1.2 Self-learning potential

DCSL-GNN utilizes a self-supervised mechanism to learn through multiple rounds. The goal is to understand the behavior of the model across these rounds under different scenarios involving noise in the network structure and node attributes. To understand the learning evolution of the model, we explore the Normalized Mutual Information (NMI) over the rounds. This metric provides insights into the learning process of DCSL-GNN and helps us determine whether the model is effectively capturing the community structure. A gradual increase in NMI values over the rounds suggests that the model is improving and refining its clustering ability with each round. To assess the self-learnin potential of DCSL-GNN, we applied it to the following scenarios:

	Attribute Means (μ)	Attribute Variance (σ^2)	Intra-Cluster	inter-cluster
Scenario 1	$[10, 20, 30, 40]$	1	0.5	0.1
Scenario 2	$[10, 20, 30, 40]$	50	0.5	0.1
Scenario 3	$[10, 20, 30, 40]$	1	0.3	0.1
Scenario 4	$[10, 20, 30, 40]$	50	0.3	0.1
Scenario 5	$[10, 20, 30, 40]$	1	0.2	0.1
Scenario 6	$[10, 20, 30, 40]$	50	0.2	0.1

Table 4.1: Indicative of each scenario of the Figure 4.1.

The experiments were performed using the following parameters:

- $N(G) = 400$, for network size;
- $K = 4$, the number of clusters;
- Two-layers GCN with dimensions $[64, 32]$ and epochs $= 100$.
- Triplet loss parameter $\Delta = 10$
- 500 random-walks with length $s = 5$, and return probability to cluster center $p = 0.5$
- Context neighborhood with $75th$ percentile visits.
- Silhouette threshold $= 0.5$ and $\alpha = 1$.

The purpose of these experiments was to assess the performance of DCSL-GNN across multiple rounds in a scenario where the clusters have equal sizes and the signals from the network and attributes are extreme (high/low network and attribute information). The results shown in Figure 4.1 demonstrate the effectiveness of the model in improving the clusters over the rounds.

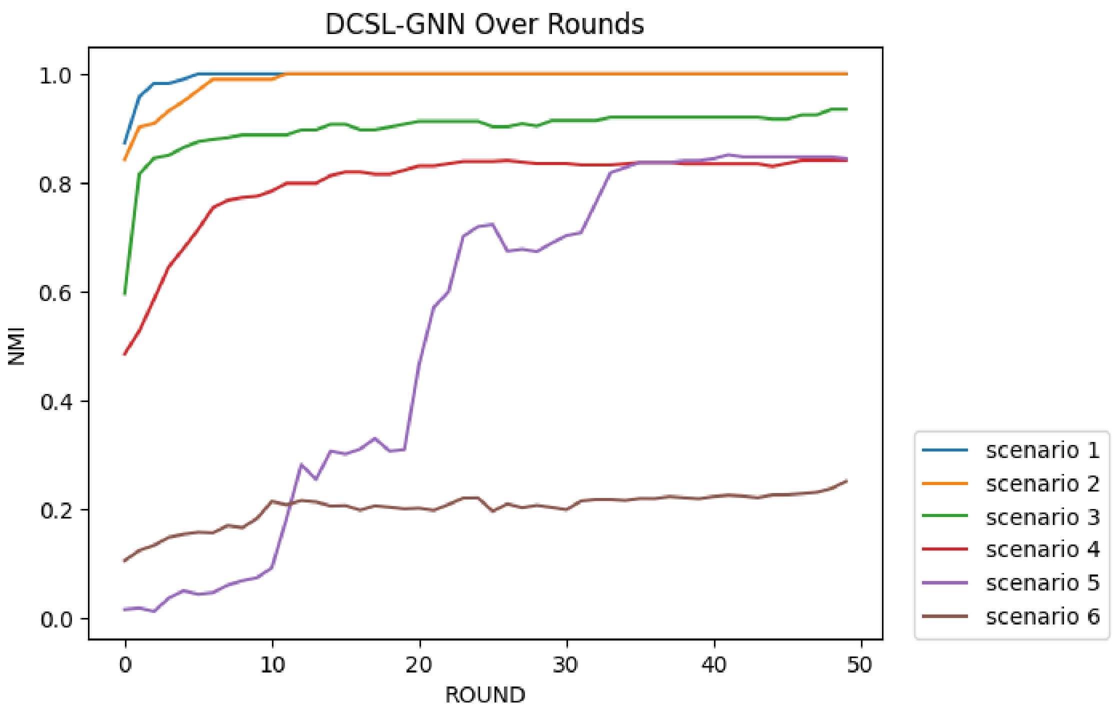
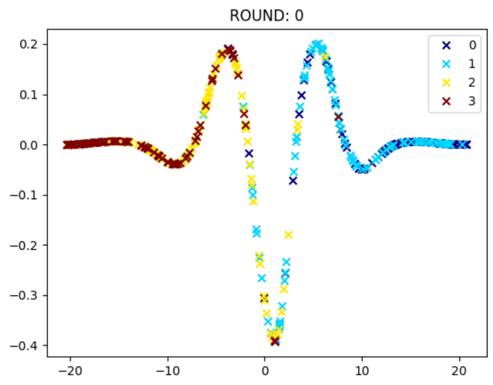


Figure 4.1: Normalized Mutual Information over rounds in multiples scenarios.

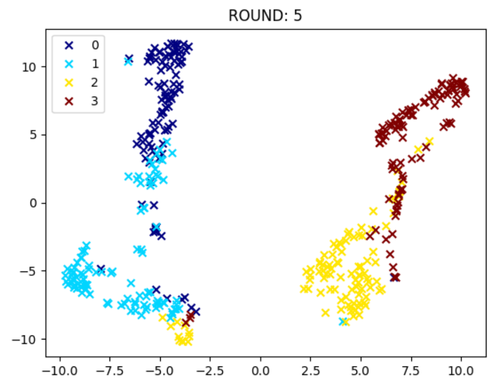
Initially, in the first few rounds, the clustering performance of DCSL-GNN is relatively low in the extreme scenarios (5 and 6). However, as the rounds progress, the model adapts and adjusts its parameters, leading to an improvement in the quality of the clusters. This improvement can be observed by analyzing the NMI values over the rounds. A lower NMI indicates less agreement between the predicted clusters and the ground truth clusters. However, as the rounds increase, the NMI values tend to increase, indicating a better alignment between the predicted and ground truth clusters. This suggests that DCSL-GNN is effectively learning and capturing the underlying community structure of the graph.

Note that after several rounds, the model reaches an upper limit and does not show further improvement (Scenario 6). This is likely due to the relatively weak signals from both the network structure and attribute distributions. The probability of connections within a cluster is only twice the probability of connections between clusters. In the graph generated, each community has 100 nodes, resulting in a higher average degree between clusters (30) compared to the average degree within clusters (20). Therefore, the model receives more information from nodes outside the cluster, leading to embeddings that are less distinct and closer to nodes from other clusters. Additionally, the attribute distributions in this scenario are very sparse, which makes it challenging for the model to capture meaningful information from them. Due to the challenges posed by the sparse attribute distribution and the relatively weak signals from the network structure, the model only achieves a maximum NMI of around 0.2.

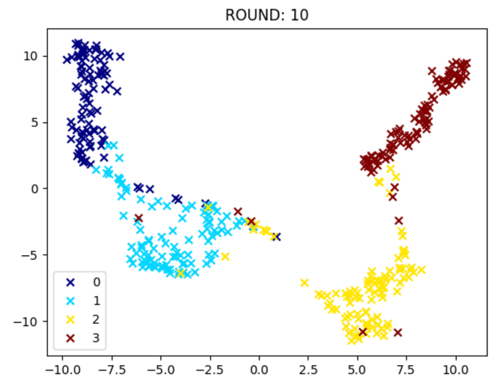
We can observe that when the signals from attributes help to indicate the community structure, the model performs better. When the intra-cluster probability is fixed and there is a lower variance of data attributes within each community, it becomes easier for the model to learn the community structure. In such cases, the attributes provide more reliable and consistent information that aligns with the underlying community divisions. In contrast, when there is a higher variance of data attributes within each community or when the attribute signals are not strongly indicative of the community structure, the model have to trust only the network information. Therefore, the quality and relevance of the attribute signals play a crucial role in the performance of the model. When the attributes provide meaningful and informative signals about the community structure, it enhances the model's ability to learn and detect the communities effectively.



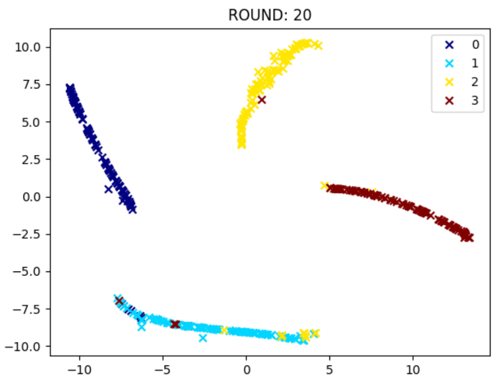
(a) Round 0



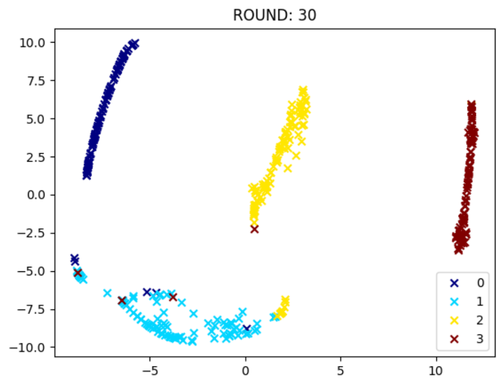
(b) Round 5



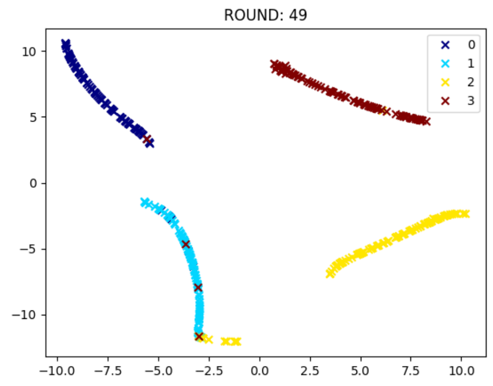
(c) Round 10



(d) Round 20

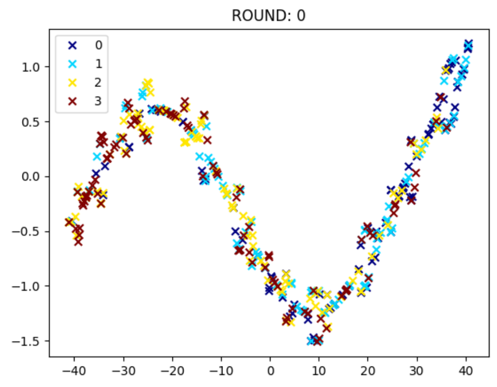


(e) Round 30

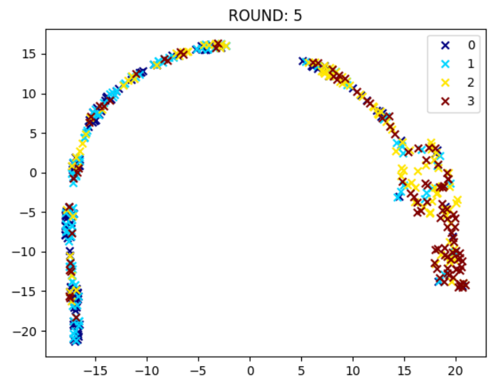


(f) Last Round 49

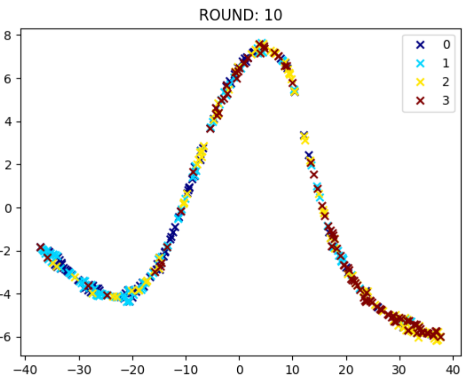
Figure 4.2: Nodes embeddings evolve over the rounds of Scenario 4, where the intra-cluster value is set to 0.3 and σ^2 is equal to 50. The embeddings are visualized using a two-dimensional projection via t-SNE.



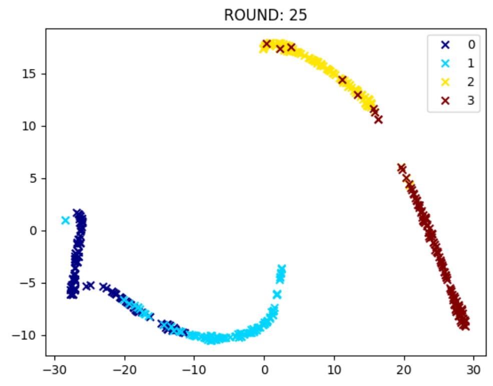
(a) Round 0



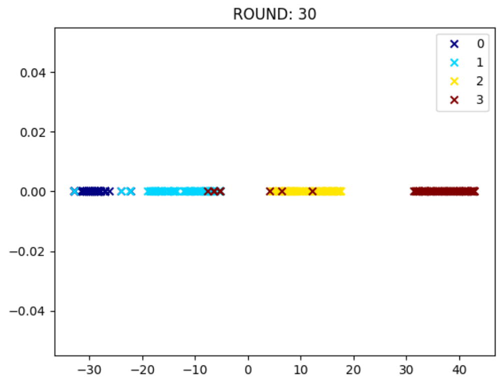
(b) Round 5



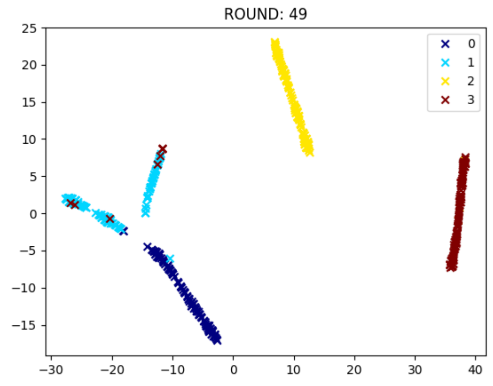
(c) Round 10



(d) Round 25



(e) Round 30



(f) Last Round 49

Figure 4.3: Nodes embeddings evolve over the rounds of Scenario 5, where the intra-cluster value is set to 0.2 and σ^2 is equal to 1. The embeddings are visualized using a two-dimensional projection via t-SNE.

In Figures 4.2 and 4.3, the t-SNE projections of the node embeddings over the rounds are shown. Initially, as the model progresses through the rounds, DCSL-GNN generates embeddings that exhibit a closer proximity for nodes belonging to the same cluster and a greater distance for nodes in different clusters. This indicates that the model is effectively learning to separate nodes based on their community membership, guided by the Triplet-Loss discussed in Chapter 3. The loss function ensures that nodes from the same community are embedded closer to each other, while nodes from different communities are embedded farther apart, and this behavior can be observed in the t-SNE projections.

As described in Chapter 3, our framework introduces a dynamic context, where the Context Graph used as input to the model is continuously evolving based on a configuration that is most representative of the cluster detection task. Consequently, the node neighborhoods are constantly changing as the Context Graph configuration evolves with each round. To assess the stability of the model, it is crucial to verify whether the node neighborhoods are reaching a stable configuration. To measure this, we utilize the Jaccard Similarity $J(\cdot)$, which quantifies the similarity between two sets X and Y.

$$J(X,Y) = \frac{X \bigcap Y}{X \cup Y} \tag{4.3}$$

So, to evaluate the stability of the neighborhood configuration, we calculate the Mean Jaccard Similarity between the neighborhood of the current round and the neighborhood of the previous round. A high mean Jaccard Similarity, closer to 1, indicates that the neighborhoods are relatively stable, with minimal changes between rounds. On the other hand, a low mean Jaccard similarity, closer to 0, suggests that the neighborhoods are undergoing significant changes between rounds, indicating a lack of stability in the configuration.

Figure 4.4 depicts the evolution of the Mean Jaccard Similarity towards stability in the scenario where the network information is relatively stronger (intra-cluster ≥ 0.3). However, in scenarios 5 and 6, where the network information is relatively low, the Mean Jaccard Similarity remains relatively low. The stable neighborhood configuration only reaches around 30% to 40%, indicating that more than half of the neighbors change from one round to the next. This suggests that the neighborhood configuration is less stable in these scenarios due to the weaker network information.

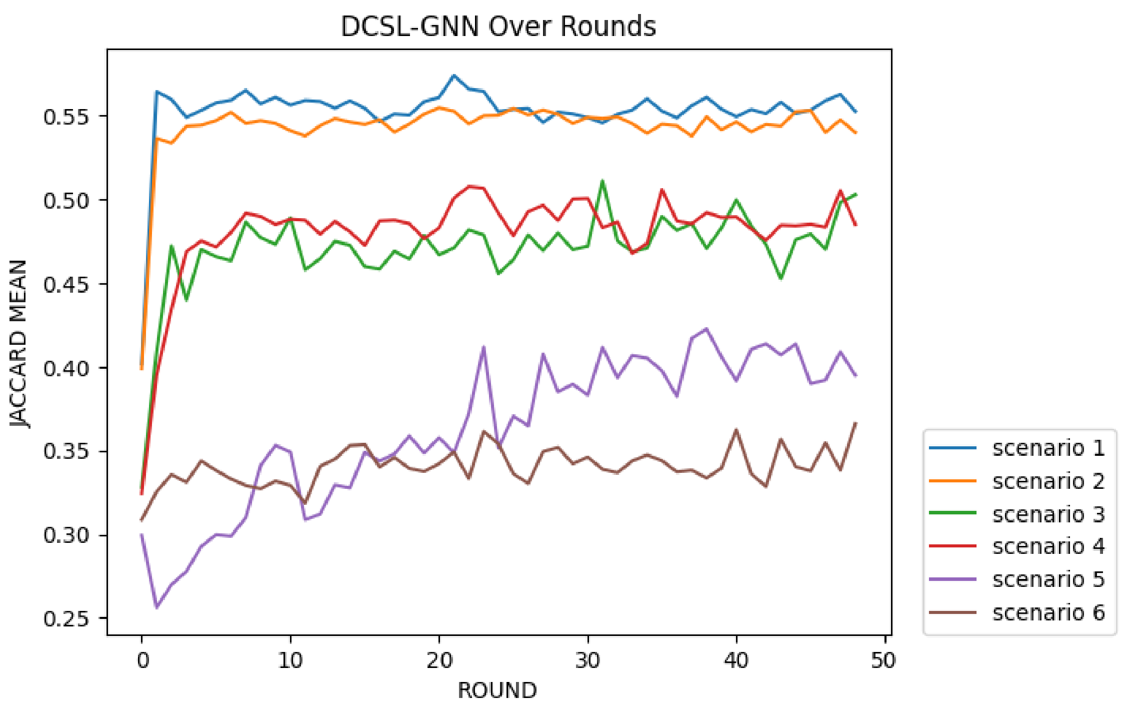


Figure 4.4: Jaccard Mean measure capturing the neighborhood stability between the current and previous rounds.

4.1.3 Network and Attributes Sensibility

This experiment aims to analyze the sensitivity of the model to varying signals from both the network structure and the attributes. To compare the results, we employ two algorithms: K-Means, which considers only the attribute signals, and Louvain, which considers only the network structure and optimizes the modularity. Additionally, to compare the effectiveness of the self-learning module, we employ DCSL-GNN with only one round, but the GCN is trained with the number of epochs of each round multiplied by the total number of rounds. We refer to this approach as NSL (not self-learning).

To ensure statistical significance given the stochastic nature of DCSL-GNN and the synthetic networks, multiple independent experiments using different sampled networks were conducted for each scenario. The reported results are the mean values obtained from the multiples runs (50 runs). This allows us to obtain more robust and reliable results by considering multiple runs and capturing the variability in the performance of the model.

Fixed Attributes Variance

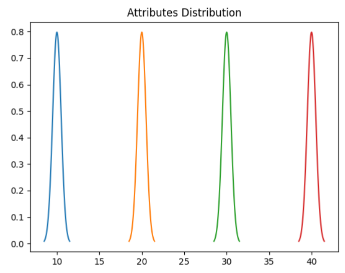
Figure 4.6 presents the results of the experiments where the variance of the attributes is fixed, while the intra-cluster probability parameter of the Stochastic Block Model (SBM) is varied. This allows us to examine the influence of individual signals on the cluster detection process. By manipulating the intra-cluster probability parameter, we can control the strength of the network structure signal and observe its impact on the quality of the detected clusters. Figure 4.5 illustrates the distribution sampled in this experiment for the attributes. The attributes were sampled from a normal distribution with a fixed mean, while the variances were varied. It can be observed that as the variance increases, the distribution curves become wider, indicating a higher spread of attribute values. This increased variance can lead to attribute data points overlapping more, which reduces the distinctiveness and importance of the attributes for the clustering problem.

It is worth noting that in Figure 4.6, the blue line represents the DCSL-GNN with supervised learning (SL), the orange dashed line represents the DCSL-GNN without supervised learning (NSL), the green dashed line represents the Louvain algorithm that considers only the network signal, and the red dashed line represents the K-means algorithm that considers only the attribute strengths. The lines represent the mean performance of 50 independent experiments, while the shaded regions represent the 25th and 75th percentiles. This provides a visualization of the variability and distribution of the results across multiple runs.

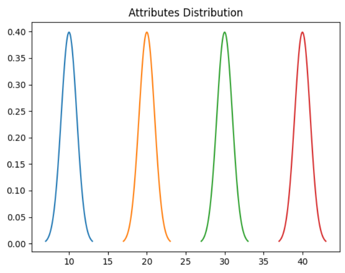
The self-learning module shows promising results, as indicated by the comparison between the blue and orange lines. In all experiments, the performance of the model with the self-learning module (blue line) is significantly higher than without it (orange line). This suggests that the self-learning module significantly contributes to the improvement of the clustering performance in the DCSL-GNN framework.

In scenarios where the attribute signal is extremely strong and effectively reveal the underlying clustering, the DCSL-GNN performance tends to be higher than algorithms which consider only the network structure (Comparation of Figures 4.6 (a) and (b)). This is because the attributes provide sufficient information to accurately determine the cluster membership of nodes. In such cases, even with weaker signals from the network structure (SBM intra-cluster probability ≤ 0.3), by combining the information from both the network and attributes, DCSL-GNN effectively capture and utilize the strong attribute signals for clustering, better than algorithms which rely solely on the network structure or attributes (Louvain and K-means algorithms).

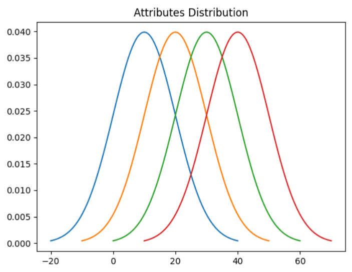
From Figure 4.6(c) and (d), it can be observed that DCSL-GNN achieves similar performance to the Louvain algorithm, even when the attribute signals are not strong. This highlights the adaptability of DCSL-GNN in leveraging different sources of information based on their relative signals strengths. However, when the attribute signals are too noisy, they can interfere and lead to lower cluster quality, as demonstrated in Figure 4.6(e).



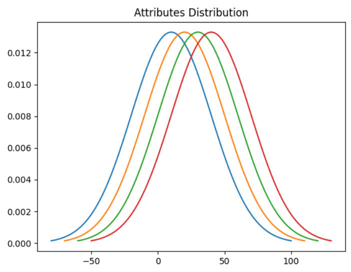
(a) Variance = 0.5



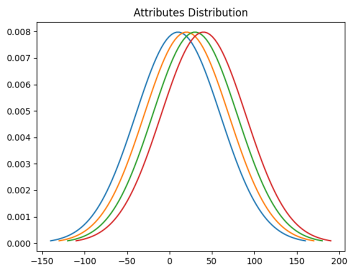
(b) Variance = 1



(c) Variance = 10

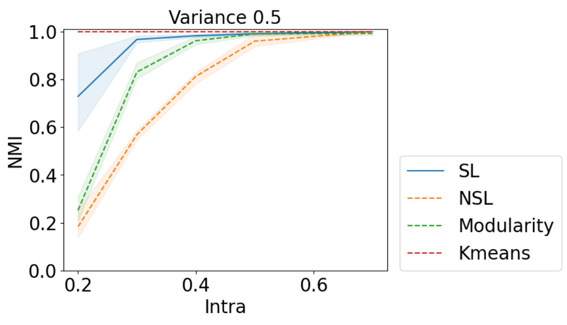


(d) Variance = 30

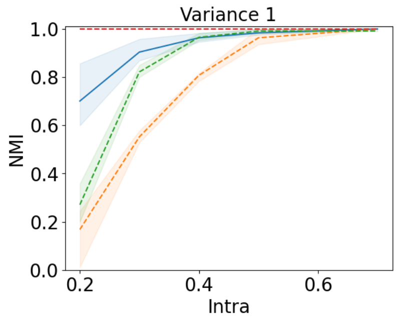


(e) Variance = 50

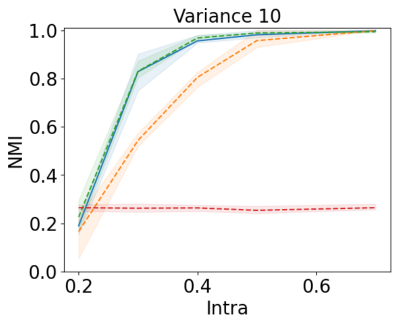
Figure 4.5: The cluster attributes in the experiments were sampled from Normal Distribution, as described in Section 2.3.



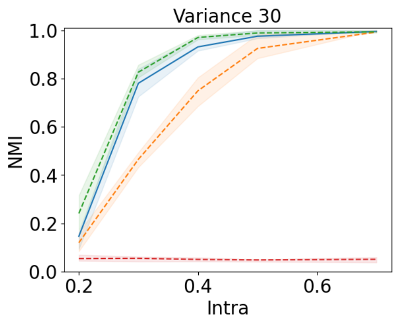
(a)



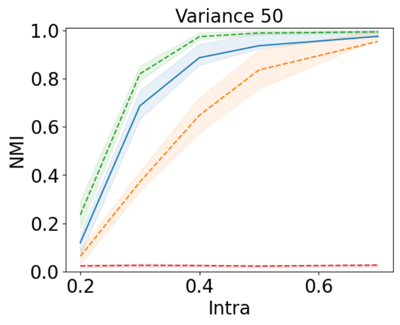
(b)



(c)



(d)



(e)

Figure 4.6: The results of the 50 runs for each experiment demonstrate that DCSL-GNN performs well in scenarios where there is low confidence in the network structure community and higher confidence in the attribute community. In these scenarios, DCSL-GNN effectively utilizes the attribute signals to compensate for the weaker network structure signals.

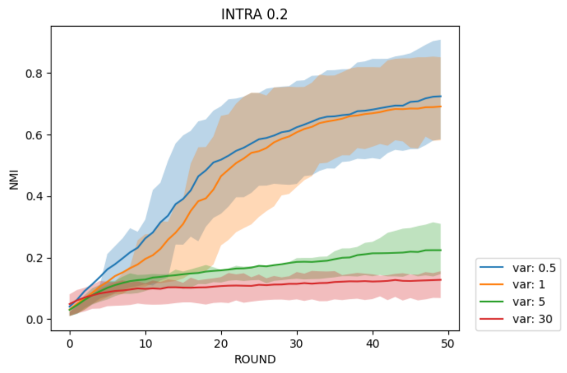
Fixed SBM Intra-cluster

This experiment evaluates the performance of DCSL-GNN with a fixed intra-cluster probability over the rounds. Figure 4.7 shows the learning process over multiple runs, where the line represents the mean and the shaded area represents the 25th and 75th percentiles. It can be observed that when the cluster signal from the network structure is not strong, the framework takes a certain number of rounds to converge. This can be seen in Figure 4.7(a), where the blue and orange lines (representing a strong attribute signal) continue to increase in value even in the last rounds. The convergence can be analyzed in Figure 4.7(b), where the attribute signals are not noisy, and the node neighborhoods in the Context Graph tend to be more consistent across rounds. However, when there are fewer network and attribute signals available, the Jaccard similarity between the neighborhood of the current and previous rounds remains low, indicating that the context neighborhood does not converge to a stable configuration.

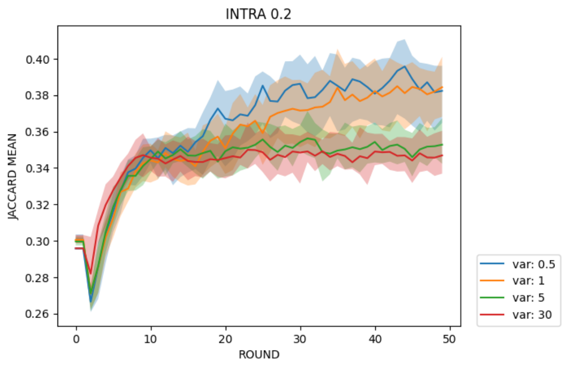
When comparing the increase in network signals, it can be observed that the Jaccard similarity increases (Figures 4.7(b), (c), and (d)). This suggests that a strong network signal contributes to a more stable neighborhood in the Context Graph. Additionally, it is worth noting that the attribute signals also have an influence on the stability. When the attribute signals are noisy, the Jaccard Similarity tends to be lower.

In Figure 4.8, we can observe different scenarios where the intra-cluster probability remains fixed, but the normal variance of the attributes is varied. It is evident that as more noise is added to the attributes, the clustering results are negatively influenced. However, when the network structure is strong enough, the influence of attribute signals becomes less significant to DCSL-GNN. In such cases, the inherent connectivity in the network itself are sufficient to reveal the underlying clustering patterns. Note that in these scenarios, Louvain algorithm (green line) achieves the highest NMI score among the evaluated algorithms in Figure 4.6(c), (d), (e) and (f).

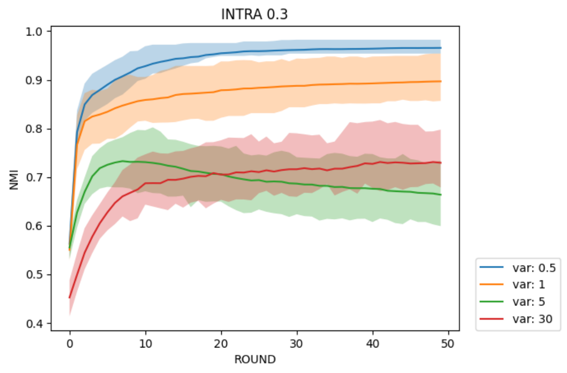
Note that the K-means, which consider only the attributes, when the variance is high, and are considered just noisy, because none information can be extracted.



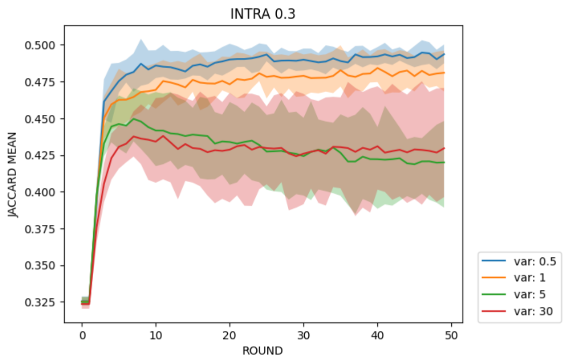
(a)



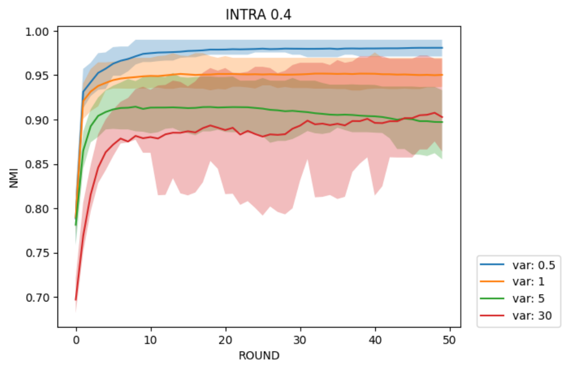
(b)



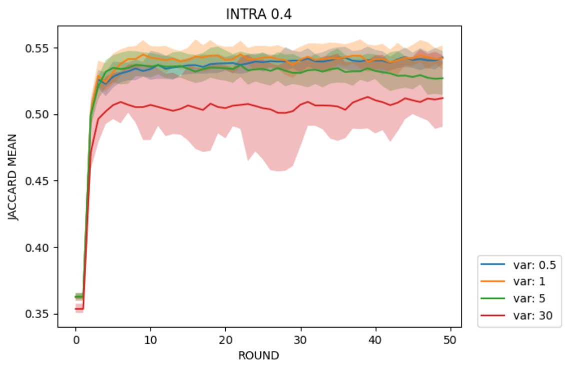
(c)



(d)

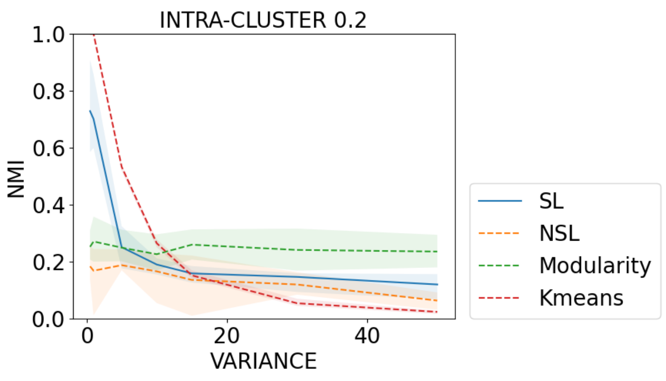


(e)

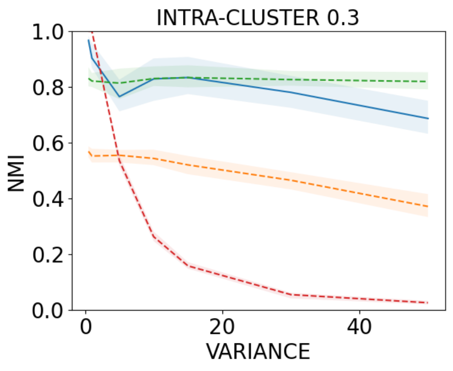


(f)

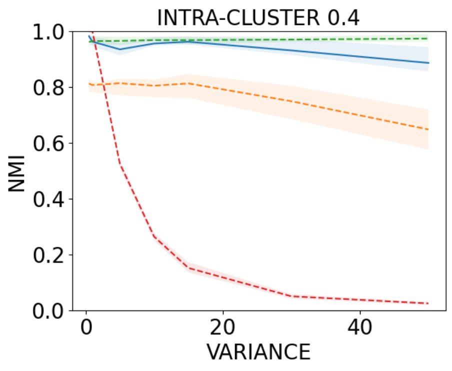
Figure 4.7: DCSL-GNN evolution over rounds, with fixed intra-cluster probability and varying attribute signals. Results include NMI over rounds and the Jaccard mean of the neighborhood between the current and previous rounds.



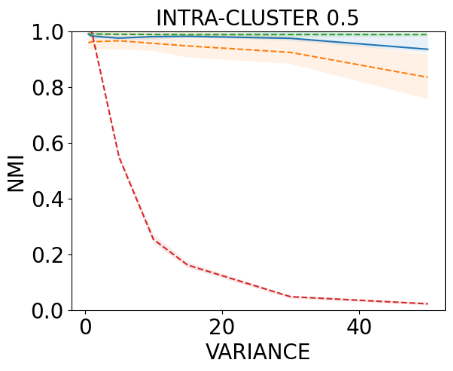
(a)



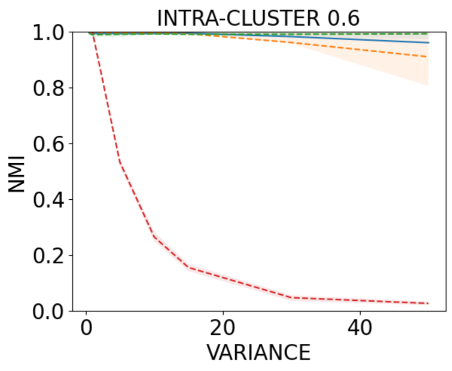
(b)



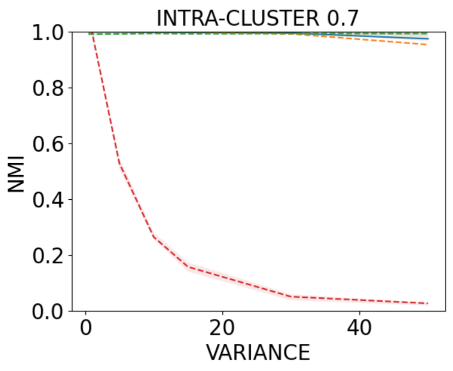
(c)



(d)



(e)



(f)

Figure 4.8: DCSL-GNN performance with fixed intra-cluster probability and varying attribute signals.

4.2 Benchmarks datasets

We conducted experiments on three citation networks: Cora, CiteSeer, and PubMed. These datasets are widely used to test network analysis algorithms, particularly in community detection studies, where they serve as benchmark graphs to evaluate the performance of various frameworks and methods, as discussed in Chapter 2. DCSL-GNN was executed 20 times with manually tuned parameters specific to each dataset.

4.2.1 Datasets Overview

The Cora dataset, introduced by Andrew McCallum et al.[62], consists of 2708 scientific publications classified into seven classes. The dataset also includes 5278 links representing the citations between the publications. Each publication in the dataset is described by a binary word vector indicating the absence or presence of each word from a dictionary of 1433 unique words. The Citeseer dataset, introduced by Lee Giles et al.[63], is also a collection of scientific publications. Similar to the Cora dataset, the features in Citeseer are represented by vectors that encode the presence or absence of words from a dictionary. The Pubmed dataset, introduced by Prithviraj Sen et al.[64], consists of scientific publications from the PubMed database related to diabetes. The publications are classified into three classes. The features for each publication are represented by TF/IDF weighted word vectors derived from a dictionary of 500 unique words. The basic information of these datasets is shown in Table4.2.

Dataset	Nodes	Edges	F	K	cc	$< k >$	Imbalance
Cora	2708	5278	1433	7	0.24	3.89	4.54
Citeseer	4230	5337	602	6	0.11	2.52	1.48
PubMed	19717	44324	500	3	0.06	4.49	1.91

Table 4.2: Network characteristics of the benchmark datasets. F represents the dimension of the node attributes, K is number of clusters, cc denotes the average clustering coefficient of the network, $< k >$ is the average node degree, and imbalance is the ratio between the largest and smallest cluster.

Imbalanced class sizes pose challenges to machine learning algorithms, and Graph Neural Networks (GNNs) are no exception [65, 66]. When the majority class dominates the dataset, the loss function can be biased towards that class, leading to poor performance. The class sizes of the citation benchmarks are described in Table 4.3. Observe that the Cora dataset exhibits the highest level of class imbalance, with the largest class (Class 4) having more than 4.5 times the number of examples compared to the smallest class (Class 7). Furthermore, the size of the largest class is more than 2 times the mean class size. The most balanced dataset is Citeseer, with well-distributed class sizes and sizes spread out around the class mean.

Sizes	Cora	Citeseer	PubMed
Class 1	351	628	4103
Class 2	217	740	7739
Class 3	418	778	7875
Class 4	818	831	/
Class 5	426	558	/
Class 6	298	695	/
Class 7	180	/	/
Total	2708	4230	19717

Table 4.3: Each row corresponds to the total number of nodes that belong to each class. The "/" symbol is used to indicate a missing class in the dataset.

4.2.2 Results

Table 4.4 presents the evaluation of multiple frameworks for clustering attributed graphs, with the NMI metric used to measure the clusters quality. It is noteworthy that for each benchmark dataset, there is one algorithm that outperforms the others. In the Cora dataset, the MGCCN framework [28] achieves the highest NMI value and is considered state-of-the-art. Similarly, in the PubMed dataset, SEComm [7] is considered the state-of-the-art. Furthermore, the proposed framework DCSL-GNN demonstrates state-of-the-art performance in the Citeseer dataset.

	Cora	Citeseer	PubMed
Kmeans	12.8 ± 3.5	26.54 ± 2.3	26.77 ± 0.2
Louvain	41.49	28.62	21.81
GAE	40.69	18.34	22.97
GIC	53.70	45.30	31.90
AGC	53.68	41.13	31.59
MGCCN	60.20	43.20	–
DNENC-Att	52.80	39.70	26.60
DeMoN	48.80	33.70	29.80
CommDGI	57.90	41.90	35.70
SEComm	56.04	42.53	36.50
DCSL-GNN	46.95 ± 3.1	47.74 ± 2.8	24.27 ± 1.20

Table 4.4: Normalized Mutual Information (NMI) results on Cora, Citeseer and PubMed datasets. Highlighted in red is the best result for each dataset.

These results demonstrate that DCSL-GNN performs well in scenarios where the clusters are balanced, both in synthetic and real data. Specifically, the performance on the Citeseer dataset provides validation for this hypothesis. However, it is worth noting that imbalanced nodes can affect the performance of DCSL-GNN. This is because the triplet-loss used for learning the model parameters relies on the selection of triplets consisting of an anchor, positive, and negative node. Thus, the classes with fewer examples contribute less to the objective function, which can result in the dominant class biasing the model. This phenomenon was observed in the Cora dataset, where the model is biased by the largest cluster, leading to lower performance of DCSL-GNN in this scenario.

Chapter 5

Conclusion and Future work

Graph clustering in attributed graphs has indeed emerged as a prominent research area, garnering significant attention, especially with the growing popularity of Graph Neural Networks (GNNs). GNNs have the ability to capture both structural and attribute information, making them suitable for graph clustering tasks. Their application in graph clustering has demonstrated promising results, as they can effectively leverage both types of information to achieve more accurate and effective clustering compared to methods that consider only one of them. Sereral tecniques that have been developed utilizing GNNs to trackle the graph clustering task was discussess [6, 7, 12, 27, 47–49]

In this dissertation, we propose a novel fully unsupervised framework for clustering network attributes. Specifically, our framework combines Graph Neural Networks (GNNs) with a self-supervised module that iteratively constructs improved node representations based on the cluster assignments. The GNN component leverages the structural and attributes information in the network to capture meaningful representations which are utilized in the clustering process. The self-supervised module operates iteratively, using the cluster assignments obtained in previous iterations to guide the learning process. It takes advantage of the refined node representations at each iteration to update and improve the cluster assignments. This iterative approach enables the framework to refine both the node representations and the clustering assignments in a mutually beneficial manner.

The evaluation of the proposed framework takes into account various scenarios where the signals of node attributes and network structures are varied. This approach allows for a comprehensive assessment of the framework performance across different conditions and signal strengths. A synthetic graph generator was designed to simulate different combinations of attribute signals and network structures. By manipulating the strength of attribute signals and the connectivity of the network, we can create diverse scenarios that represent a range of real-world conditions.

In addition to the synthetic graphs, real datasets commonly used as benchmarks in the research community were also employed in the evaluation. These real datasets provide a means to validate the framework performance in real-world scenarios and compare it against existing methods. We observed that the proposed model achieved state-of-the-art performance in one of the benchmark datasets that were utilized.These results demonstrate the effectiveness and competitiveness of the proposed model in tackling the graph clustering problem.

However, it is worth noting that the model performance varied across different datasets. In some cases, the model exhibited lower performance compared to other methods in some cases. This discrepancy can be attributed to the specific characteristics of the datasets, such as variations in network structure, attribute signals, presence of noise or outliers and imbalanced classes.

5.1 Future work

There are several directions for further development that are worth pursuing:

- Compare DCSL-GNN with other frameworks in the same synthetic scenarios. Evaluate the clustering performance and identify cases where DCSL-GNN outperforms or falls behind compared to other approaches. This comparison can provide insights into the relative strengths and limitations of DCSL-GNN.
- Improving the scalability of the proposed model, particularly in terms of memory load on the GPU, is an important aspect to address. As the iterative process constructs a new graph at each iteration, loading these new graphs into GPU memory can lead to a memory bottleneck and impact the overall scalability of the model.
- Investigate the impact of imbalanced class distributions on the model, and measuring this impact is crucial to understand the challenges posed by class imbalance.
- Many frameworks combine multiple loss functions to optimize their performance. Adding another signal learning component, especially one that utilizes an objective function focused on modularity, can be beneficial.
- Design and implement different optimizations to DCSL-GNN such that it can handle very large graphs (e.g., millions of nodes and edges) while maintaining low training effort and high accuracy in the clustering process.

References

[1] BARABÁSI, A.-L., PÓSFAI, M. *Network science*. Cambridge, Cambridge University Press, 2016. ISBN: 9781107076266 1107076269. Disponível em: <`http://barabasi.com/networksciencebook/`>.

[2] PEROZZI, B., AL-RFOU, R., SKIENA, S. "Deepwalk: Online learning of social representations". In: *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, pp. 701–710, 2014.

[3] RIBEIRO, L. F., SAVERESE, P. H., FIGUEIREDO, D. R. "struc2vec: Learning node representations from structural identity". In: *Proceedings of the 23rd ACM SIGKDD international conference on knowledge discovery and data mining*, pp. 385–394, 2017.

[4] BRONSTEIN, M. M., BRUNA, J., COHEN, T., et al. "Geometric deep learning: Grids, groups, graphs, geodesics, and gauges", *arXiv preprint arXiv:2104.13478*, 2021.

[5] HAMILTON, W., YING, Z., LESKOVEC, J. "Inductive representation learning on large graphs", *Advances in neural information processing systems*, v. 30, 2017.

[6] WANG, C., PAN, S., CELINA, P. Y., et al. "Deep neighbor-aware embedding for node clustering in attributed graphs", *Pattern Recognition*, v. 122, pp. 108230, 2022.

[7] BANDYOPADHYAY, S., PETER, V. "Unsupervised constrained community detection via self-expressive graph neural network". In: *Uncertainty in Artificial Intelligence*, pp. 1078–1088. PMLR, 2021.

[8] GAN, G., MA, C., WU, J. *Data clustering: theory, algorithms, and applications*. SIAM, 2020.

[9] DESHPANDE, Y., SEN, S., MONTANARI, A., et al. "Contextual stochastic block models", *Advances in Neural Information Processing Systems*, v. 31, 2018.

[10] STANLEY, N., BONACCI, T., KWITT, R., et al. "Stochastic block models with multiple continuous attributes", *Applied Network Science*, v. 4, n. 1, pp. 1–22, 2019.

[11] ZHANG, X., LIU, H., WU, X.-M., et al. "Spectral embedding network for attributed graph clustering", *Neural Networks*, v. 142, pp. 388–396, 2021.

[12] MÜLLER, E. "Graph clustering with graph neural networks", *Journal of Machine Learning Research*, v. 24, pp. 1–21, 2023.

[13] JAIN, A. K., MURTY, M. N., FLYNN, P. J. "Data clustering: a review", *ACM computing surveys (CSUR)*, v. 31, n. 3, pp. 264–323, 1999.

[14] SIBSON, R. "SLINK: An optimally efficient algorithm for the single-link cluster method", *The Computer Journal*, v. 16, n. 1, pp. 30–34, 01 1973. ISSN: 0010-4620. doi: 10.1093/comjnl/16.1.30. Disponível em: <`https://doi.org/10.1093/comjnl/16.1.30`>.

[15] DEFAYS, D. "An efficient algorithm for a complete link method", *The Computer Journal*, v. 20, n. 4, pp. 364–366, 01 1977. ISSN: 0010-4620. doi: 10.1093/comjnl/20.4.364. Disponível em: <`https://doi.org/10.1093/comjnl/20.4.364`>.

[16] JAIN, A. K. "Data clustering: 50 years beyond K-means", *Pattern recognition letters*, v. 31, n. 8, pp. 651–666, 2010.

[17] FORTUNATO, S. "Community detection in graphs", *Physics reports*, v. 486, n. 3-5, pp. 75–174, 2010.

[18] KERNIGHAN, B. W., LIN, S. "An efficient heuristic procedure for partitioning graphs", *The Bell system technical journal*, v. 49, n. 2, pp. 291–307, 1970.

[19] BARNES, E. R. "An algorithm for partitioning the nodes of a graph", *SIAM Journal on Algebraic Discrete Methods*, v. 3, n. 4, pp. 541–550, 1982.

[20] NEWMAN, M. E., GIRVAN, M. "Finding and evaluating community structure in networks", *Physical review E*, v. 69, n. 2, pp. 026113, 2004.

[21] BLONDEL, V. D., GUILLAUME, J.-L., LAMBIOTTE, R., et al. "Fast unfolding of communities in large networks", *Journal of Statistical Mechanics: Theory and Experiment*, v. 2008, n. 10, pp. P10008, oct 2008. doi: 10.1088/1742-5468/2008/10/P10008. Disponível em: <`https://dx.doi.org/10.1088/1742-5468/2008/10/P10008`>.

[22] LANCICHINETTI, A., FORTUNATO, S., KERTÉSZ, J. "Detecting the overlapping and hierarchical community structure in complex networks", *New journal of physics*, v. 11, n. 3, pp. 033015, 2009.

[23] DUCH, J., ARENAS, A. "Community detection in complex networks using extremal optimization", *Physical review E*, v. 72, n. 2, pp. 027104, 2005.

[24] FORTUNATO, S., BARTHELEMY, M. "Resolution limit in community detection", *Proceedings of the national academy of sciences*, v. 104, n. 1, pp. 36–41, 2007.

[25] HOLLAND, P. W., LASKEY, K. B., LEINHARDT, S. "Stochastic blockmodels: First steps", *Social Networks*, v. 5, n. 2, pp. 109–137, 1983. ISSN: 0378-8733. doi: https://doi.org/10.1016/0378-8733(83)90021-7. Disponível em: <`https://www.sciencedirect.com/science/article/pii/0378873383900217`>.

[26] CHAMI, I., ABU-EL-HAIJA, S., PEROZZI, B., et al. "Machine Learning on Graphs: A Model and Comprehensive Taxonomy", *Journal of Machine Learning Research*, v. 23, n. 89, pp. 1–64, 2022. Disponível em: <`http://jmlr.org/papers/v23/20-852.html`>.

[27] HE, D., SONG, Y., JIN, D., et al. "Community-centric graph convolutional network for unsupervised community detection". In: *Proceedings of the twenty-ninth international conference on international joint conferences on artificial intelligence*, pp. 3515–3521, 2021.

[28] LIU, L., KANG, Z., RUAN, J., et al. "Multilayer graph contrastive clustering network", *Information Sciences*, v. 613, pp. 256–267, 2022.

[29] ZHANG, T., XIONG, Y., ZHANG, J., et al. "CommDGI: community detection oriented deep graph infomax". In: *Proceedings of the 29th ACM international conference on information & knowledge management*, pp. 1843–1852, 2020.

[30] ZHOU, J., CUI, G., HU, S., et al. "Graph neural networks: A review of methods and applications", *AI open*, v. 1, pp. 57–81, 2020.

[31] WU, Z., PAN, S., CHEN, F., et al. "A comprehensive survey on graph neural networks", *IEEE transactions on neural networks and learning systems*, v. 32, n. 1, pp. 4–24, 2020.

[32] PALOWITCH, J., TSITSULIN, A., PEROZZI, B., et al. "Synthetic Graph Generation to Benchmark Graph Learning". In: *NeurIPS 2022 Workshop: New Frontiers in Graph Learning*.

[33] MAEKAWA, S., SASAKI, Y., FLETCHER, G., et al. "GenCAT: Generating attributed graphs with controlled relationships between classes, attributes, and topology", *Information Systems*, v. 115, pp. 102195, 2023. ISSN: 0306-4379. doi: https://doi.org/10.1016/j.is.2023.102195. Disponível em: <`https://www.sciencedirect.com/science/article/pii/S0306437923000315`>.

[34] YING, R., HE, R., CHEN, K., et al. "Graph convolutional neural networks for web-scale recommender systems". In: *Proceedings of the 24th ACM SIGKDD international conference on knowledge discovery & data mining*, pp. 974–983, 2018.

[35] BRONSTEIN, M. M., BRUNA, J., LECUN, Y., et al. "Geometric Deep Learning: Going beyond Euclidean data", *IEEE Signal Processing Magazine*, v. 34, n. 4, pp. 18–42, 2017. doi: 10.1109/MSP.2017.2693418.

[36] LECUN, Y., BOTTOU, L., BENGIO, Y., et al. "Gradient-based learning applied to document recognition", *Proceedings of the IEEE*, v. 86, n. 11, pp. 2278–2324, 1998.

[37] YU, Y., SI, X., HU, C., et al. "A review of recurrent neural networks: LSTM cells and network architectures", *Neural computation*, v. 31, n. 7, pp. 1235–1270, 2019.

[38] MIKOLOV, T., SUTSKEVER, I., CHEN, K., et al. "Distributed representations of words and phrases and their compositionality", *Advances in neural information processing systems*, v. 26, 2013.

[39] GROVER, A., LESKOVEC, J. "node2vec: Scalable feature learning for networks". In: *Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining*, pp. 855–864, 2016.

[40] HAMILTON, W. L. "Graph representation learning", *Synthesis Lectures on Artifical Intelligence and Machine Learning*, v. 14, n. 3, pp. 1–159, 2020.

[41] HAMILTON, W. L., YING, R., LESKOVEC, J. "Representation learning on graphs: Methods and applications", *arXiv preprint arXiv:1709.05584*, 2017.

[42] KIPF, T. N., WELLING, M. “Semi-supervised classification with graph convolutional networks”, *arXiv preprint arXiv:1609.02907*, 2016.

[43] XIAO, S., WANG, S., DAI, Y., et al. “Graph neural networks in node classification: survey and evaluation”, *Machine Vision and Applications*, v. 33, pp. 1–19, 2022.

[44] CAI, L., LI, J., WANG, J., et al. “Line Graph Neural Networks for Link Prediction”, *IEEE Transactions on Pattern Analysis and Machine Intelligence*, v. 44, n. 9, pp. 5103–5113, 2022. doi: 10.1109/TPAMI.2021.3080635.

[45] DWIVEDI, V. P., JOSHI, C. K., LAURENT, T., et al. “Benchmarking graph neural networks”, 2020.

[46] SU, X., XUE, S., LIU, F., et al. “A comprehensive survey on community detection with deep learning”, *IEEE Transactions on Neural Networks and Learning Systems*, 2022.

[47] WANG, X., LI, J., YANG, L., et al. “Unsupervised learning for community detection in attributed networks based on graph convolutional network”, *Neurocomputing*, v. 456, pp. 147–155, 2021.

[48] CUI, G., ZHOU, J., YANG, C., et al. “Adaptive graph encoder for attributed graph embedding”. In: *Proceedings of the 26th ACM SIGKDD international conference on knowledge discovery & data mining*, pp. 976–985, 2020.

[49] HOU, Z., LIU, X., CEN, Y., et al. “Graphmae: Self-supervised masked graph autoencoders”. In: *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pp. 594–604, 2022.

[50] BARCELÓ, P., KOSTYLEV, E. V., MONET, M., et al. “The Logical Expressiveness of Graph Neural Networks”. In: *International Conference on Learning Representations*, 2020. Disponível em: <`https://openreview.net/forum?id=r1lZ7AEKvB`>.

[51] BALCILAR, M., RENTON, G., HÉROUX, P., et al. “Analyzing the Expressive Power of Graph Neural Networks in a Spectral Perspective”. In: *International Conference on Learning Representations*, 2021. Disponível em: <`https://openreview.net/forum?id=-qh0M9XWxnv`>.

[52] LI, Q., HAN, Z., WU, X.-M. “Deeper insights into graph convolutional networks for semi-supervised learning”. In: *Thirty-Second AAAI conference on artificial intelligence*, 2018.

[53] ROUSSEEUW, P. J. “Silhouettes: a graphical aid to the interpretation and validation of cluster analysis”, *Journal of computational and applied mathematics*, v. 20, pp. 53–65, 1987.

[54] NG, A., JORDAN, M., WEISS, Y. “On spectral clustering: Analysis and an algorithm”, *Advances in neural information processing systems*, v. 14, 2001.

[55] NIELSEN, F., NIELSEN, F. “Hierarchical clustering”, *Introduction to HPC with MPI for Data Science*, pp. 195–211, 2016.

[56] HARTIGAN, J. A., WONG, M. A., OTHERS. “A k-means clustering algorithm”, *Applied statistics*, v. 28, n. 1, pp. 100–108, 1979.

[57] DEMPSTER, A. P., LAIRD, N. M., RUBIN, D. B. “Maximum likelihood from incomplete data via the EM algorithm”, *Journal of the royal statistical society: series B (methodological)*, v. 39, n. 1, pp. 1–22, 1977.

[58] MURPHY, K. P. *Probabilistic machine learning: an introduction.* MIT press, 2022.

[59] Bengio, Y., LeCun, Y. (Eds.). *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015. Disponível em: <`https://iclr.cc/archive/www/doku.php%3Fid=iclr2015:accepted-main.html`>.

[60] REZA, F. M. *An introduction to information theory.* Courier Corporation, 1994.

[61] SHANNON, C. E. “A mathematical theory of communication”, *The Bell system technical journal*, v. 27, n. 3, pp. 379–423, 1948.

[62] MCCALLUM, A. K., NIGAM, K., RENNIE, J., et al. “Automating the construction of internet portals with machine learning”, *Information Retrieval*, v. 3, pp. 127–163, 2000.

[63] GILES, C. L., BOLLACKER, K. D., LAWRENCE, S. “CiteSeer: An automatic citation indexing system”. In: *Proceedings of the third ACM conference on Digital libraries*, pp. 89–98, 1998.

[64] SEN, P., NAMATA, G., BILGIC, M., et al. “Collective classification in network data”, *AI magazine*, v. 29, n. 3, pp. 93–93, 2008.

[65] ZHAO, T., ZHANG, X., WANG, S. "Graphsmote: Imbalanced node classification on graphs with graph neural networks". In: *Proceedings of the 14th ACM international conference on web search and data mining*, pp. 833–841, 2021.

[66] WANG, Y., ZHAO, Y., SHAH, N., et al. "Imbalanced graph classification via graph-of-graph neural networks". In: *Proceedings of the 31st ACM International Conference on Information & Knowledge Management*, pp. 2067–2076, 2022.