MASSACHUSETTS INSTITUTE OF TECHNOLOGY

NETWORK SCIENCE AND MODELS

6.268

Comparing Graph Clustering Methods

Authors:
Filip Ryzner
Mohammad Mustafa Arif

May 15, 2022

Comparing Graph Clustering Methods

Filip Ryzner, Mohammad Mustafa Arif

Abstract—Graph clustering (community detection) is an extremely useful tool in scientific inquiry and data analytics as communities identify agents representing similar properties, distinct from the rest of the network. Classical methods of community detection such as Spectral Clustering have been successful at identifying such communities, but more recently, newer methods have been proposed with more favourable asymptotic run-time complexities. In this paper, we compare Spectral Clustering with Louvain's Algorithm, another classical method, along with three deep learning approaches for community detection: GraphEncoder, MinCutPool, and the Structural Deep Convolutional Network (SDCN). We present the theoretical framework behind these techniques along with an asymptotic complexity analysis. Finally, we run an experiment on synthetically generated graphs from the Stochastic Block Model to evaluate the performance of these methods by the normalised mutual information criteria. We find that in the absence of non graphical features, the GraphEncoder and MinCutPool are unable to achieve performance comparable to classical methods for our graphs. We do find, however, that SDCN, which combines building blocks from GraphEncoder and MinCutPool, beats Louvain's Algorithm on all our experiments. This promising insight motivates the application for deep learning as a problem solving philosophy on graphical data, especially given its ability to combine graphical and non-graphical information in a task specific manner. The code for all our experimental work is available at https://github.com/arifmoh2/community-detetction-deep-learning.

1 Introduction

The ultimate goal of clustering algorithms is to place object with similar properties into the same groups [Nascimento and De Carvalho, 2011]. Oftentimes the clustering algorithms are applied to graph networks as these structures arise commonly in a wide variety of domains including but not limited to sociology - community detection [Blondel et al., 2008], biology - gene expression analysis [Huttenhower et al., 2007], and computer science with applications in natural language processing [Ushioda, 1996] as well as image segmentation [Bianchi et al., 2020], [Malliaros and Vazirgiannis, 2013]. This results in a family of clustering algorithms falling in the boroad category of called 'graph clustering'. Moreover, graph clustering algorithms can be applied even to datasets, which themselves are not structured as graph networks in their original form, such as the Wine Dataset, which was introduced in [Cortez et al., 2009]. In these cases, an algorithm approximating the graph structure of the data is first utilized, before the augmented dataset in a graph representation is fed into a graph clustering algorithm. An example of one of the most common graph structure approximating algorithms is the K-nearest neighbours algorithm [Fix and Hodges, 1989].

However, clustering in general, thus including graph clustering, is an unsupervised learning problem, therefore algorithms are not trained using the information about the true distribution of community labels as these are only used for the algorithm evaluation. Given the broad range of applications of graph clustering combined with the increasing availability of data across various disciplines, it is of high importance to use graph clustering algorithms that can not only perform well on datasets with various structure (e.g. clustering coefficient, average node degree, graph diameter), but do also scale well in terms of compute time (i.e. directly related to computational complexity) when the size of the graph increases.

However, the currently popular graph clustering methods, namely Spectral Clustering [Von Luxburg, 2007] and the Louvain's Method [Blondel et al., 2008], are relatively computationally expensive with asymptotic complexities of $\Theta(n^{2.333})$ and $\Theta(nlog(n))$ respectively. This poses significant scalability issues in terms for large graphs with thousands or even millions of nodes. Therefore, given the concurrent increase in the computational power availability, researchers have started to look into applying deep learning approaches to the graph clustering problem.

The goal of our work is to provide the reader with an understanding of the currently popular graph clustering methods (both deep-learning methods as well as non-deep learning methods) and inspect their performance in specific scenarios. Specifically, in our work we will first present several graph clustering algorithms, which we will then compare on a handpicked set of test graphs generated synthetically according to predefined specifications. Our main objective is to compare the state-of-the-art algorithms on the same datasets such that performance across the algorithms can be compared. We aim to do this as we have noticed that most of the academic literature deriving new promising approaches do not test on the same datasets and do not compare against other existing methods.

The rest of this paper is structured in the following way. In the Chapter 2, we provide a brief literature review mapping out the recent advances in the field of graph clustering. Next, in the Chapter 3, we describe all the methods that will be analysed. The Chapter 3 is split into two sections, the Non-Deep Learning Methods

and Deep Learning Methods. Next, in the Chapter 4, we provide a description of the datasets used for testing. Chapter 5 presents the results of our analysis and finally in Chapter 6, we summarise our findings, conclude, and present potential directions for future investigation.

2 Literature review

In this section we provide a review of the literature related to the graph clustering, including literature specifically focusing on reviewing approaches used for the community detection problem. As our work mostly focused on exploring the newly arising deep learning approaches to the graph clustering problem, we will put substantial emphasis on discussing literature concerned with application of deep learning approaches to this problem.

Non-deep learning clustering algorithms remain popular to date, not only because they are often simple to understand and their logic is transparent, unlike the black-box deep learning algorithms, but more importantly because they often offer a stable and predictable performance. They have been studied for a long time and their behaviour across various scenarios is understood better than in the case of the arising deep learning approaches. The classical example of a clustering algorithm applied to graph problems, oftentimes community detection, is the Spectral Clustering, described in [Ng et al., 2001]. This method relies on Eigen-Value decomposition (EVD) of the graph similarity matrix, a process with an asymptotic runtime complexity of is $\Theta(n^{2.333})$, which significantly bottlenecks the scalability of the overall algorithm. [White and Smyth, 2005] have shown that the spectral relaxation of the Q function maximisation used in advanced graph clustering method first introduced by [Newman and Girvan, 2004], results in an algorithm which scales almost linearly for large sparse networks.

A newer and better scalable graph clustering algorithm, famously used for community detection, is Louvain's Algorithm [Blondel et al., 2008]. This greedy, two stage algorithm has computational complexity of $\Theta(nlog(n))$ and aims to maximise the graph modularity metric, first proposed by [Newman and Girvan, 2004], which measures the strength of division of a network into modules. Several improvements of the Louvain's Algorithm has been since introduced. One example is the Louvain+ algorithm, introduced by [Gach and Hao, 2013], which generalized the [Blondel et al., 2008] by including an uncoarsening phase that results in a full multi-level method, and demonstrated the positive impact it has on the performance of the initial algorithm. Another example is the [De Meo et al., 2011], which exploited a novel measure of edge centrality, based on the $k - paths$, definition available in [Bresar et al., 2011] and showed that this approach has a positive impact on the performance of the original version of the Louvain's Algorithm for community detection.

With the successful adoption of deep learning across a variety of fields, it has started to gain traction in applications related to the graph clustering. The first application of deep-learning to graph clustering was introduced in [Tian et al., 2014] and was called the Graph Encoder. The Graph Encoder first learns a non-linear graph embedding using stacked shallow autoencoders, introduced by [Vincent et al., 2010], and then uses the k-means clustering algorithm [Lloyd, 1982] on vertex embeddings obtained from the autoencoder to produce the resulting clustering. Moreover, [Tian et al., 2014] demonstrated that their approach is theoretically similar to Spectral Clustering [Von Luxburg, 2007], but has a significantly lower computational complexity, claiming $\Theta(n)$ on large sparse graphs. Later approaches, such as [Xie et al., 2016] and [Yang et al., 2017], have further improved and optimized the combination of stacked autoencoders with acompaigned by the k-means clustering algorithm and obtained an increase in the clustering accuracy.

However, it has been pointed out, that the graph embedding obtained using the autoencoders mostly encode information about the global structure of the graph (higher order information), while the local structure (lower order information) is not well captured, [Bo et al., 2020a]. To address this problem, approaches utilising Graph Neural Networks (GNN), reviewed in [Zhou et al., 2020], have been developed and successfully applied. One such application is the MinCut-Pooling layer in deep GNNs from [Bianchi et al., 2020], who formulate a continuous relaxation of the normalized minimum cut (min-cut) problem, defined in [Bianchi et al., 2019], and train a GNN to produce cluster assignments that minimize the relaxed objective. The authors provide theoretical illustration of the similarity between this approach and the Spectral Clustering and demonstrate a performance improvement in a variety of applications. Next, the [Tsitsulin et al., 2020] derive another graph pooling based approach, this time optimizing the network modularity, which they have shown to significantly outperform the min-cut based approach of [Bianchi et al., 2019]. Lastly, we mention the work by [Bo et al., 2020a], which combines both GNNs and Autoencoders to effectively encode low and high order structural information, claiming a significant improvement in performance.

3 Methodology

3.1 Pre-requisities

In this section we briefly introduce some standard preliminary notation and algorithms that are used in the following section.

3.1.1 *K-means clustering*

The K-means algorithm introduced by [Lloyd, 1982], [MacQueen et al., 1967] is one of the foundational clustering algorithms. The algorithm starts with random

cluster initialisation where k cluster centers are randomly selected from available datapoints and all other datapoints are then assigned to their closest corresponding cluster center based on the utilised distance metric, such as the Euclidean distance. Cluster center is then updated based on selected update policy and points are re-assigned. This process repeats until the convergence criterion is met. The algorithm can be formulated as in Algorithm 1:

Algorithm 1 K-means Clustering

Variables:
$\mathcal{T} = \{x_i\}_{i=1}^{N}$ observation set; K - cluster count
$\{C_k\}_{k=1}^{K}$ - cluster centres; $\{\mathcal{T}_k\}_{k=1}^{K}$ - data clusters
Algorithm:
1. Initialise cluster centres: $\{c_k\}_{k=1}^{K} \sim \mathcal{U}(\mathcal{T})$
2. Assignment optimisation (closest cluster centre):
$\mathcal{T}_k = \left\{ \mathbf{x} \in \mathcal{T} : \forall j, \|\mathbf{x} - \mathbf{c}_k\|^2 \leq \|\mathbf{x} - \mathbf{c}_j\|^2 \right\} \quad \forall k$
3. Cluster centre optimisation (mean value):
$c_k = \begin{cases} \frac{1}{|\mathcal{T}_k|} \sum_{x \in \mathcal{T}_k} x & \text{if} \quad |\mathcal{T}_k| > 0 \\ \text{re-initialize,} & \text{if} \quad \mathcal{T}_k = \emptyset \end{cases}$
4. Terminate if $\forall k : \mathcal{T}_{k,t+1} = \mathcal{T}_{k,t}$; else go to 2

The upside of the K-means algorithm is it always converges as it either minimises the distance metric across observations from their respective cluster centres by changing the cluster assignments or does not change the assignments, thus converging to a solution. Therefore, the algorithm can never repeat two same assignments in the process. However, the k-means algorithm is not guaranteed to converge to the global minimum and its convergence properties depend heavily on it's initialisation, which in the base algorithm is random [Bottou and Bengio, 1994].

Therefore, to address the initialisation issues, in our implementations we always use the the k-means++ algorithms introduced by [Arthur and Vassilvitskii, 2006], which ensures a smarter initialisation of the centroids and improves the quality of the clustering, while also making the algorithm less prone to volatile performance across runs stemming from uniformly random initialisation.

3.1.2 Graph Convolutional Networks (GCNs)

Early stage, perceptron style neural networks utilised regular Euclidean data as network inputs and optimized networks for general learning tasks. However, much of the real world data has underlying graph structures [Malliaros and Vazirgiannis, 2013], motivating the need for neural networks that could accommodate and exploit this additional information. One variant of such Neural networks are Graph Convolutions Neural Networks (GCNs) originally proposed in 2017 [Kipf and Welling, 2017].

The fundamental idea behind GCNs is that in addition to specific features corresponding to individuals (vertices), they also take as an input information regarding the inter-connectivity amongst individuals (edges). This is achieved by passing in the adjacency matrix at every forward propagation step in the network. Thus, in deep neural networks, hidden representations for the graph can be learnt based on node features as well as inter-connectivity. Specifically, the forward pass in a Graph Convolutional Network is defined as follows:

$$H^{[i+1]} = \sigma(W^{[i]} H^{[i]} A^*) \tag{1}$$

Here, $H^{[i]}$ corresponds to the hidden layer state in the $i-th$ layer of the network, A^* to the normalized [1] adjacency matrix, $W^{[i]}$ represents trainable weights for the layer i, and $\sigma(.)$ is a non-linear activation function (e.g. sigmoid activation as described in [Nwankpa et al., 2018]).

Finally, the hidden representations resulting from a deep GCN can be used as a dense representation of the input data that accounts for features and graph structure simultaneously, and it can be fed into a final task specific block (such as a perceptron followed by a softmax activation function for classification). The whole network can then be trained end-to-end via backpropagation.

3.2 Non-Deep Learning Methods

In this section we introduce and review the two non-deep learning approaches that we will be using in our work; namely Spectral Clustering and Louvain's Algorithm. While both these algorithms are also applicable to non-graph based datasets, we limit our attention to the formulation assuming that our problem is based on graph representation $G = (V, E)$, where V is the set of graph vertices and E is the set of edges in the graph.

3.2.1 Spectral Clustering

Spectral Clustering is a well known clustering algorithm documented across many publications. We are basing the following description and analysis on [Von Luxburg, 2007] and [Tian et al., 2014]. The logic is formulated in Algorithm 2.

Algorithm 2 Spectral Clustering

1. Compute the similarity matrix $\mathbf{S}$ based on $\mathbf{A}$
2. Compute the Laplacian matrix: $\mathbf{L}$ = $\mathbf{D}$ - $\mathbf{S}$
3. Normalize the Laplacian: $\mathbf{L}_{norm}$ = $\mathbf{D}^{-\frac{1}{2}}\mathbf{L}\mathbf{D}^{-\frac{1}{2}}$
4. Eigen-Value decomposition of $\mathbf{L}_{norm} = \mathbf{Q}\mathbf{\Lambda}\mathbf{Q}^{-1}$
5. Take top K eigenvectors from $\mathbf{Q}$ into $\mathbf{H}$
6. Cluster rows of $\mathbf{H}$ using K-means

where $\mathbf{A}$ is the adjacency matrix of graph G, the similarity matrix $\mathbf{S}$ is calculated based on selected similarity metric (e.g. cosine similarity, description available in [Lahitani et al., 2016]), $\mathbf{Q}$ is the eigen-vector matrix and $\mathbf{\Lambda}$ is the diagonal matrix of eigen-values. Finally,

1. Normalization of the adjacency matrix by the diagonal degree matrix is needed to make the scale of the feature vectors invariant as explained in [Kipf and Welling, 2016].

as $\mathbf{L_{norm}}$ is also symmetric and its rows correspond to observations.

However, due to the worse than quadratic complexity of the Eigen-Value Decomposition EVD ($O(n^{2.333})$) [Tian et al., 2014] step, Spectral Clustering does not scale well for very large graphs. Moreover, for large-sized graphs, the second eigenvalue of the normalised Laplacian matrix is often ill-conditioned, leading to slow convergence of iterative eigenvalue solvers, thus further decreasing the performance [Tian et al., 2014].

3.2.2 Louvain's Algorithm

Louvain's method for community detection [Blondel et al., 2008] is a greedy, bottom up, clustering algorithm that optimises the modularity of the network as defined by Equation 2:

$$Q = \frac{1}{2m}\sum_{i,j}[A_{ij} - \frac{k_i k_j}{2m}]\delta(c_i, c_j) \tag{2}$$

where A_{ij} represents the edge weight between nodes i and j; k_i and k_j are the sum of the weights of edges attached to nodes i and j respectively; m is the sum of all edge weights in the graph; c_i and c_j are the communities of the nodes; and δ is the Kronecker delta function.

The algorithm operates via a two stage approach. During the first stage, each node forms its own community, and then for each node i, a change in modularity is computed by removing i from its own community and putting it into the community of each neighbour $j \in$ neighbours(i). Node i is then placed into the community that maximises the increase in modularity, and if no increase is possible, it is kept in its original community. The process is applied repeatedly until a local maxima is achieved. In the second stage, all nodes in the same community are represented by a single node and a new graph is built by essentially collapsing the initial one inwards, then the first step is applied again so that effectively, clusters are merged together iteratively according to the modularity maximisation criteria. The pseudocode is summarised in the Algorithm 3.

One advantage of Louvain's algorithm is that the number of clusters is not pre-specified; the algorithm continues to merge clusters together until further merging to increase modularity is not possible. In addition, the algorithm has a computational complexity of $\Theta(nlog(n))$ making it asymptotically more efficient than Spectral Clustering. However, since the algorithm is myopic in nature, optimality is only guaranteed in a local sense.

Algorithm 3 Louvain's Algorithm

```
initialise G = Initial Network
while TRUE do
    Put each node of G in its own community
    while some nodes are moved do
        for all nodes i of G do
            place i in its neighbouring community (including its own) which maximises modularity gain as defined by Equation 2.
        end for
    end while
    if total modularity > initial modularity then
        G = network between communities of G
    else
        Terminate
    end if
end while
```

3.3 Deep Learning Methods

In this section we discuss three deep learning approaches that we will be using in our experiments. In chronological order of publication, these approaches are the GraphEncoder[Tian et al., 2014], the MinCutPool[Bianchi et al., 2019], and the Structural Deep Clustering Network (SDCN)[Bo et al., 2020b]. It should be noted that MinCutPool and SDCN differs from the GraphEncoder and the previously discussed non-deep learning approaches in a fundamental way: they cluster based on a network structure as well as individual node features. The following subsections will describe these methods in detail.

3.3.1 Graph Encoder

[Tian et al., 2014] presented one of the first approaches to utilise deep learning for graph clustering via an autoencoder based representation learning approach. For the sake of brevity, this method will be referred to as a 'Graph Encoder' in this paper.

The Graph Encoder is built upon a building block of a simple 3-layer autoencoder that attempts to reconstruct an input by passing it through a lower dimensional hidden layer with non-linear activation functions.

The fundamental intuition behind the vanilla autoencoder is that it optimises the task of reconstructing the input, thereby making the middle lower dimensional layer an efficient representation of the input. As such, it is effectively performing dimensionality reduction [2]. The Graph Encoder architecture builds upon the autoencoder by stacking them together to obtain dense, low dimensional embeddings for the nodes in the graph. Effectively, the input to the next autoencoder is the hidden layer representation of the previous autoencoder, as shown in Figure 1.

Finally, k-means clustering is performed on the the hidden layer representation outputted by the last autoencoder in the stacked architecture. The whole process is summarised in the Algorithm 4.

Each stacked autoencoder is trained to minimize the following loss function:

$$Loss(\theta) = \sum_{i=1}^{n} ||y_i - x_i||_2 + \beta * KL(\rho||\hat{\rho}) \tag{3}$$

2. In this regard, it is very similar to Spectral Clustering, where we take a rank-k approximation of the normalised Laplacian to reduce problem dimensionality

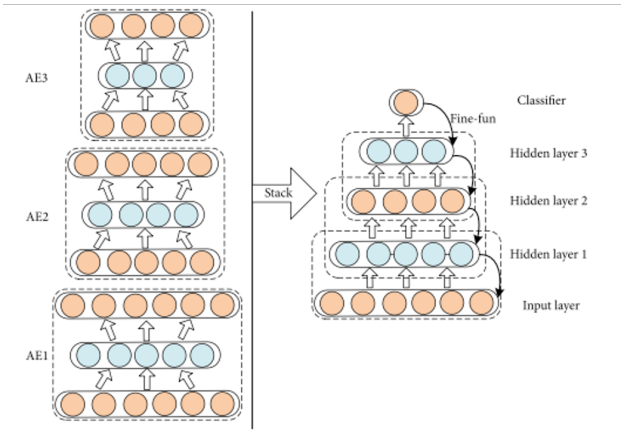


Fig. 1: Stacked Autoencoder, [Liu et al., 2018]

Algorithm 4 GraphEncoder Training Algorithm

input
n-node graph G with similarity matrix $S \in \mathbb{R}^{n \times n}$ and degree matrix $D = diag(d_1, ..., d_n)$ where d_i denotes the degree of node i; DNN layers number Γ, with the number of layers $n^{(j)}$ in layer j; $n^{(1)} = n$; $X^{(j)} \in \mathbb{R}^{n \times n^{(j)}}$ is the input to layer j;
initialise $X^{(1)} = D^{-1}S$
for j=1 to Γ **do**
1. Build a three layer vanilla autoencoder with input data $X^{(j)}$
2. Iteratively train the autoencoder by optimising the loss function defined in Equation 3. Obtain hidden layer activations $h^{(j)}$
3. Set $X^{(j+1)} = h^{(j)}$
end for
Run k-means on $X^{(\Gamma)} \in \mathbb{R}^{n \times n^{(\Gamma)}}$
output Final clustering results

Where, the second term is a regularization penalty that we impose in addition to the standard reconstruction loss. ρ is set to be a small constant, and $\hat{\rho} = \frac{1}{n}\sum_{j=1}^{n} h_j$ is the average of the hidden layer activations. $KL(\rho||\hat{\rho})$ divergence is non-symmetric and defined as:

$$KL(\rho||\hat{\rho}) = \sum_{j=1}^{|\hat{\rho}|} \rho log(\frac{\rho}{\hat{\rho}_j}) + (1-\rho) log(\frac{1-\rho}{1-\hat{\rho}_j}) \quad (4)$$

Note that all the optimization functions are smooth and differentiable, making iterative learning feasible. [Tian et al., 2014] analysed the algorithm from a high level perspective and compared the optimization problem to Spectral Clustering. They have proven that Spectral Clustering via truncated singular value decomposition produces the best rank-k approximation to a matrix in terms of the Frobenius norm. Thus, we can see a similarity between Spectral Clustering and Graph Encoder in the sense that both the methods aim to optimise for reconstruction error in a way that provides low-dimensional representations of the input.

However, the Graph Encoder offers one significant advantage; it has a runtime complexity of $\Theta(n * c * d)$ where d is the maximum number of hidden layer nodes, n is the number of nodes in the graph, and c is the average degree. Thus we see that Graph Encoder is linear in the number of nodes (unlike Spectral Clustering which is worse than quadratic), and it can be trained via parallalization whereas Spectral Clustering is much more difficult to parallelize [Bottou, 1999].

3.3.2 *MinCutPool*

The MinCutPool [Bianchi et al., 2019] is a graph neural network based approach that is motivated by the K-way normalised min-CUT optimisation problem. The K-way normalised minCUT partitions the graph into K-disjoint clusters through edge removal such that a minimum total volume of edges is removed from the original graph. By definition [Shi and Malik,], this can be expressed as:

$$\begin{aligned} \textbf{min } & \frac{1}{K}\sum_{k=1}^{K} \frac{links(\nu_k)}{degree(\nu_k)} \\ \textbf{min } & \frac{1}{K}\sum_{k=1}^{K} \frac{\sum_{i,j \in \nu_k} \varepsilon_{i,j}}{\sum_{i \in \nu_k, j \in \nu \setminus \nu_k} \varepsilon_{i,j}} \end{aligned} \quad (5)$$

where the numerator counts the edge volume within each cluster and the denominator counts the edged between the nodes in a cluster and the rest of the graph. [Dhillon et al., 2004] expressed this problem in matrix notation as given in the following equation:

$$\textbf{max } \frac{1}{K}\sum_{k=1}^{K} \frac{C_k^T A C_k}{C_k^T D C_k} \quad (6)$$

where $C \in \{0,1\}^{N \times K}$ corresponds to a binary cluster assignment matrix, C_k denotes the $k-th$ column of C, and we have that $C1_K = 1_N$ since every point must belong to some cluster. Note that the optimisation problem expressed in (6) is NP-hard. It is therefore recast in a relaxed continuous formulation that can be solved in polynomial time and guarantees a near optimal solution [Bianchi et al., 2019]. However, this relaxed problem also involves EVD, and therefore suffers from the same runtime complexity as Spectral Clustering.

To address these concerns, [Bianchi et al., 2019] propose a graph neural net with a MinCutPool layer to approximate the solution to (6). The approach essentially involves the use of a Graph Convolutional Network [Kipf and Welling, 2016] that embeds the nodes in the original graph into representative features and then uses a multilayer perceptron ([Gardner and Dorling, 1998]) for soft-cluster assignments. The architecture of this layer is visually summarised in the Figure 2.

Note that one essential component of this solution is that, in addition to taking simply the graph structure as an input, it also utilizes individual features corresponding to each node, denoted by X. Thus, clustering is

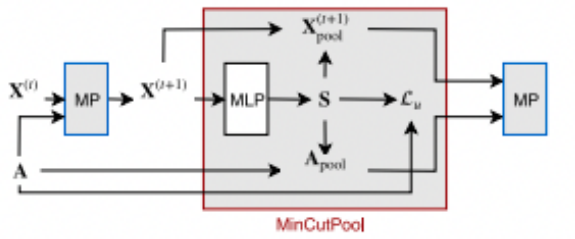


Fig. 2: Schema of the MinCutPool Layer

informed by graph structure as well as individual node features. Mathematically, the process can be expressed as in Equation 7.

$$\begin{aligned} \bar{X} &= GCN(X, \bar{A}; \Theta_{GNN}) \\ S &= MLP(\bar{X}; \Theta_{MLP}) \end{aligned} \tag{7}$$

where $\bar{A}$ denotes the normalised adjacency matrix, X denotes individual node features, and Θ_{GNN} and Θ_{MLP} denote trainable parameters. The MLP is designed with a softmax activation so that it is guaranteed that $s_{i,j} \in [0, 1]$ and $S1_K = 1_N$. The model allows the construction of deeper architectures by stacking together single units of the MinCutPool Layer from Figure 2 and applying pooling operations between the layers, as summarised in Equation 8.

$$\begin{aligned} A^{pool} &= S^T AS \\ X^{pool} &= S^T X \end{aligned} \tag{8}$$

Since A^{pool} corresponds to L_c in the loss function in (10), trace maximization yields clusters with many internal connections that are weakly connected across. Thus A^{pool} will be a diagonal dominant matrix with more prominent self connections than any other connections. To correct for this so that propagation in subsequent layers between adjacent nodes is not compromised, the following layer-wise normalizations are also needed when stacking MinCutPool layers:

$$\begin{aligned} \hat{A} &= A^{pool} - \mathbf{I_K} diag(A^{pool}) \\ \tilde{A}^{pool} &= \hat{D}^{-\frac{1}{2}} \hat{A} \hat{D}^{-\frac{1}{2}} \end{aligned} \tag{9}$$

The network is optimized according to a continuous relaxation of the original minCUT problem illustrated in (6). In particular, the loss function is defined as follows:

$$L = L_c + L_o = \frac{-Tr(S^T \bar{A} S)}{-Tr(S^T \bar{D} S)} + \| \frac{S^T S}{||S^T S||_F} - \frac{\mathbf{I_K}}{\sqrt{(K}} \|_F \tag{10}$$

Here, L_c evaluates the minCUT given the soft cluster assignment, as can be seen by comparing it to the original problem in (6). It is minimised at $L_c = -1$, corresponding to the case where the graph has K connected components and the cluster assignment is the same for all nodes in the same component and orthogonal to cluster assignments for nodes in other components. Conversely, it is maximised at $L_c = 0$, corresponding the the case where for each pair of connected nodes, the cluster assignments are orthogonal. This is visually illustrated in Figure 3.

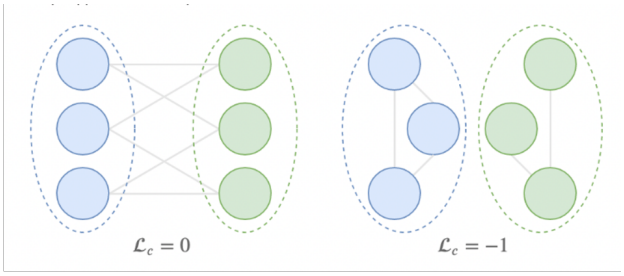


Fig. 3: Extreme Cases for L_c

However, note that since L_c is non-convex, it is prone to local minima or degenerate solutions. For instance, it is minimized when all nodes are assigned to the same cluster, or, due to the continuous cluster assignment relaxation, all nodes are uniformly assigned to all clusters. To penalize such scenarios, L_o is added as a regularization penalty to the loss function as it encourages cluster assignments to be orthogonal and clusters to be of similar size. Also note that since both the matrices in L_o have unit norm, $0 \leq L_o \leq 2$, making the scale between L_o and L_c commensurable so that they can be added together without the need for rescaling.

The space complexity of MinCutPool is $O(NK)$ since it requires storing the cluster assignments $S \in \mathbb{R}^{N \times K}$. The computational complexity is dominated by the optimization of L_c, and is $O(NK(K + N))$ in the general case and $O(K(E + NK))$ if operations are optimized for the average case, where N is the number of nodes, K is the number of clusters, and E is the number of non-zero edges in the normalized adjacency matrix. Therefore, with MinCutPool, we can optimise for the continuous relaxation of the minCUT problem presented in (6) with more favourable asymptotic complexity than that from eigendecomposition, which was required to solve it optimally. The caveat here is that since L_c is non-convex, optimality through iterative learning is not guaranteed in theory. In practise, however, the inclusion of the L_o term in the loss function disincentivezes highly sub-optimal solutions. Furthermore, MinCutPool allows for an efficient way to combine information from individual node features as well as the node interconnectivity or graph structure when forming communities, making it a versatile tool for community detection/graph clustering.

3.3.3 The Structural Deep Clustering Network

The Structural Deep Clustering Network (SDCN) introduced by [Bo et al., 2020a] is the most complex model we review. The SDCN combines the autoenconder representation learning with the GCN representation learning in an attempt to efficiently encode both the local information (low-order structure) and the global information (high-order structure). To achieve this a delivery operator is introduced, which is used to transfer the representation learned by a specific layer in the autoencoder to

a corresponding layer in the GCN. The training of these modules is lead by a dual supervised module, which is discused later in this section. The overall architecture of the network is visualised in the Figure 4. We break the architecture review process into explanation of the corresponding modules in the architecture: the Autoencoder (DNN) Module, GCN Module and the Dual Self-Supervised Module guiding the training process.

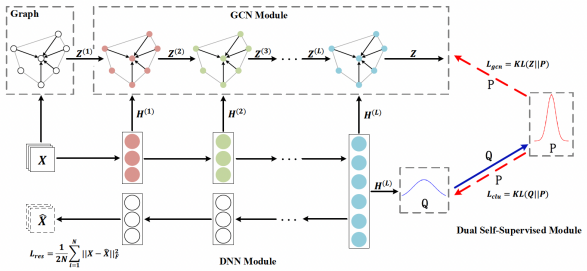


Fig. 4: SDCN architecture

DNN module:
We first review the (Deep Neural Network) DNN Module, which corresponds to the autoencoder component of the model. The DNN module comprises of a single autoencoder, which takes as an input the similarity matrix $\mathbf{S}$ of the graph G and outputs a hidden representation $\mathbf{H}^{(l)}$ from each of its layers. A hidden layer corresponding to either the Encoder or Decoder section of the autoencoder is defined as:

$$\mathbf{H}^{(\ell)} = \phi\left(\mathbf{W}^{(\ell)}\mathbf{H}^{(\ell-1)} + \mathbf{b}^{(\ell)}\right) \tag{11}$$

where ϕ is the sigmoid activation function (as described in [Nwankpa et al., 2018]), $\mathbf{W}^{(l)}$ is the weight matrix learned during the training and $\mathbf{b}^{l}$ is the bias scalar for the corresponding layer. Moreover, the autoencoder is trained by minimizing the reconstruction loss function defined as:

$$\mathcal{L}_{res} = \frac{1}{2N}\sum_{i=1}^{N}\|\mathbf{x}_i - \hat{\mathbf{x}}_i\|_2^2 = \frac{1}{2N}\|\mathbf{X} - \hat{\mathbf{X}}\|_F^2 \tag{12}$$

where $\hat{\mathbf{x}}_i$ corresponds to an individual reconstructed datapoint, $\hat{\mathbf{X}}$ is the whole reconstructed input, in this case the similarity matrix $\mathbf{S}$, and $||.||_F^2$ is the Frobenius matrix norm. Lastly, we note that the autoencoder is pretrained separately on the specific graph before it is used in the model.

GCN module:
Next, we describe the GCN module of the network. As can be seen in the Figure 4, the input of the GCN is the graph G itself. Moreover every layer l of the GCN is also fed the output $\mathbf{H}^{(l)}$ from the corresponding hidden layer of the DNN module. The convolutional operation on the GCN is defined as follows:

$$\mathbf{Z}^{(\ell)} = \phi\left(\widetilde{\mathbf{D}}^{-\frac{1}{2}}\widetilde{\mathbf{A}}\widetilde{\mathbf{D}}^{-\frac{1}{2}}\widetilde{\mathbf{Z}}^{(\ell-1)}\mathbf{W}^{(\ell-1)}\right) \tag{13}$$

where $\mathbf{Z}^{(\ell)}$ is the output of the layer $l-th$, $\widetilde{\mathbf{Z}}^{(\ell-1)}$ is the output of the $l-th$ layer of GCN combined with the autoencoder output as defined below in Equation 14, $\widetilde{\mathbf{A}} = \mathbf{A} + \mathbf{I}$ and $\widetilde{\mathbf{D}}_{ii} = \sum_j \widetilde{\mathbf{A}}_{\mathrm{ij}}$.$\mathbf{I}$ is the identity diagonal matrix of the adjacency matrix $\mathbf{A}$ accounting for the addition of the self-loop to each node.

Moreover, the output of the $l-th$ layer of the GCN is combined with the output of the $l-th$ hidden layer of the autoecoder in the following way:

$$\widetilde{\mathbf{Z}}^{(\ell-1)} = (1-\epsilon)\mathbf{Z}^{(\ell-1)} + \epsilon\mathbf{H}^{(\ell-1)} \tag{14}$$

where ϵ is the mixing weight (set to 0.5 in [Bo et al., 2020a]).

The very last layer of the GCN module serves as the multi-way classificator determining cluster membership and is defined via the softmax function as follows in Equation 15:

$$Z = \mathrm{softmax}\left(\widetilde{\mathbf{D}}^{-\frac{1}{2}}\widetilde{\mathbf{A}}\widetilde{\mathbf{D}}^{-\frac{1}{2}}\mathbf{Z}^{(L)}\mathbf{W}^{(L)}\right) \tag{15}$$

Lastly, we note that the first layer of the GCN takes as input the raw graph data $\mathbf{X}$, which is provided alongside adjacency matrix and contain whatever information is deemed to be suitable to be encoded to improve the clustering:

$$\mathbf{Z}^{(1)} = \phi\left(\widetilde{\mathbf{D}}^{-\frac{1}{2}}\widetilde{\mathbf{A}}\widetilde{\mathbf{D}}^{-\frac{1}{2}}\mathbf{X}\mathbf{W}^{(1)}\right) \tag{16}$$

Self-Supervised Module:
Finally, we introduce the dual self-supervised module (DSSM), which guides the training of the joint network architecture. The goal of the DSSM is to provide a unified training framework for both the GCN and the DNN since in their initial form, neither are meant to be used for clustering.

We begin by describing how the training process of the pre-trained autoencoder is done via the DSSM. The output of the middle hidden layer of the autoencoder is taken and cluster assignment of individual samples is determined via the student t-distribution (described in [Van der Maaten and Hinton, 2008]), which is used as a similarity measure as follows:

$$q_{ij} = \frac{\left(1 + \left\|\mathbf{h}_i - \boldsymbol{\mu}_j\right\|^2/v\right)^{-\frac{v+1}{2}}}{\sum_{j'}\left(1 + \left\|\mathbf{h}_i - \boldsymbol{\mu}_{j'}\right\|^2/v\right)^{-\frac{v+1}{2}}} \tag{17}$$

where i refers to the sample number, j refers to the $j-th$ cluster, $\mathbf{h_i} \in \mathbf{H}^{(\ell)}$ and μ_j is a cluster center initialized via K-means clustering on a representation learned by the pre-trained autoencoder[3], and v is the number of degrees of freedom in the student-t distribution. Essentially, q_{ij} represents the probability of assigning sample i to the $j-th$ cluster.

Using the previous results, we can calculate the target distribution P, which aims to improve the cluster cohesion as follows:

$$p_{ij} = \frac{q_{ij}^2/f_j}{\sum_{j'} q_{ij'}^2/f_{j'}} \tag{18}$$

3. Authors note: this process is essentially the same as introduced in the Graph Encoder, but here it serves only as a pre-processing step

where $f_j = \sum_i q_{i,j}$ are the soft cluster frequencies. Finally, we optimize this by minimizing the KL-divergence loss between Q and P distributions, which can help the DNN module learn a better representation for the clustering task in question by essentially making the embeddings surround the initial K-means cluster better. The KL divergence objective is formulated as:

$$\mathcal{L}_{clu} = KL(P\|Q) = \sum_i \sum_j p_{ij} \log \frac{p_{ij}}{q_{ij}} \tag{19}$$

Secondly, training the GCN module is based on the final output Z from the GCN, which can be assumed to be a cluster assignment distribution. We can again use the previously defined P distribution to guide the training of the GCN via minimization of the KL divergence objective formulated as follows:

$$\mathcal{L}_{gcn} = KL(P\|Z) = \sum_i \sum_j p_{ij} \log \frac{p_{ij}}{z_{ij}} \tag{20}$$

Finally, the overall loss function for the SDCN is formulated as follows:

$$\mathcal{L} = \mathcal{L}_{res} + \alpha \mathcal{L}_{clu} + \beta \mathcal{L}_{gcn} \tag{21}$$

where all the losses have been defined in the above sections, $\alpha > 0$ is a hyperparameter determining the trade-off between local structure preservation and cluster assignment optimization, $\beta > 0$ controls the disturbance of GCN module to the embedding space. The main benefit of the DSSM is that both the DNN and GCN are trained together, thus their results should be more cohesive.

Finally, after training for a specified number of epochs, we use the cluster assignment distribution Z from to GCN to determine final cluster assignments for all samples i as follows:

$$r_i = \arg\max_j \quad z_{ij} \tag{22}$$

4 Experiments

In the previous section we introduced two Non-Deep Learning methods and three Deep-Learning methods. In this section we will describe the datasets and experimental setup used to evaluate these methods.

We evaluated our models on a standard set of synthetically constructed data sets. Comparison across models was done via a single performance metric: the Normalized Mutual Information (NMI) score [Danon et al., 2005]. This metric was chosen as it is standard in literature and can compare partitions even with different number of clusters. As a reminder, the normalized NMI is a variant of Mutual Information which accounts for the 'amount of information' one can extract by observing one distribution regarding a second one. In our experiments, we are interested in the mutual information between the community assignments from the aforementioned algorithms and the the true community labels. For this discrete variable case, the mutual information between distribution X and Y is defined as [4]:

$$I(X;Y) = \sum_{y \in Y} \sum_{x \in X} p_{X,Y}(x,y) \log\left(\frac{p_{X,Y}(x,y)}{p_X(x)p_Y(y)}\right) \tag{23}$$

The NMI is simply a rescaled version of mutual information by the sum of the entropy of the two distributions:

$$NMI(X;Y) = \frac{2 \times I(X;Y)}{H(X) + H(Y)} \tag{24}$$

where $H(.)$ denotes the entropy. With this rescaling we are guaranteed that $0 \leq NMI \leq 1$, and thus we can compare NMI scores across different partitions, different models, and different datasets.

Our experimental work involved the generation of random graphs via the stochastic block model [Holland et al., 1983]. The stochastic block model is a generative model for random graphs where the following specification is defined:

- the number of vertices n
- a partition of the vertex set into r disjoint communities denoted C_1,...,C_r
- a symmetric probability matrix $P \in \mathbb{R}^{r \times r}$ of edge probabilities.

The random graph is constructed as follows: any two vertices $u \in C_i$ and $v \in C_j$ are connected by an undirected edge with probability P_{ij}. In our experiments, we further simplified this by predefining a single probability for the existence of an edge between nodes within any community, denoted p, and another (smaller) probability for the existence of edges between nodes across communities, denoted q. Thus, we expressed the probability matrix P as in Equation 25:

$$P = p \cdot \mathbf{I_r} + q \cdot (1 - \mathbf{I_r}) \tag{25}$$

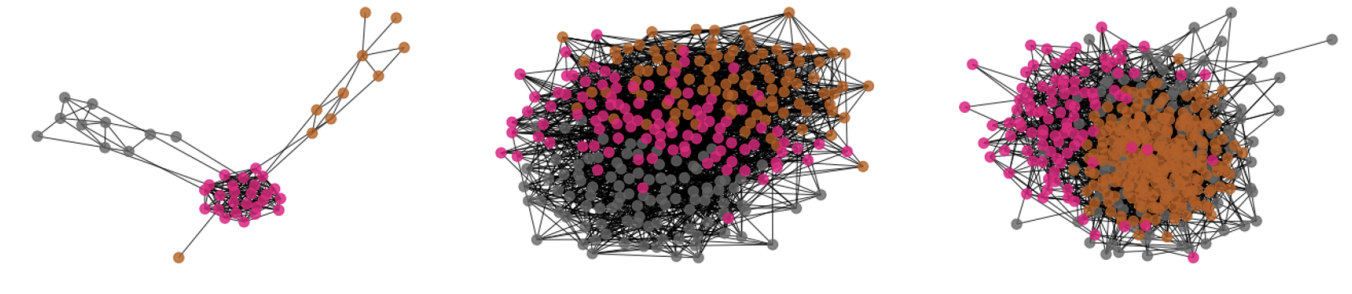
With this simplification, we generated 3 different random graphs with specifications shown in Table 1.

	Simple	Moderate	Complex
n	45	300	450
Cluster Sizes	10, 10, 25	100, 100, 100	100, 100, 250
p	0.5	0.1	0.05
q	0.01	0.02	0.01

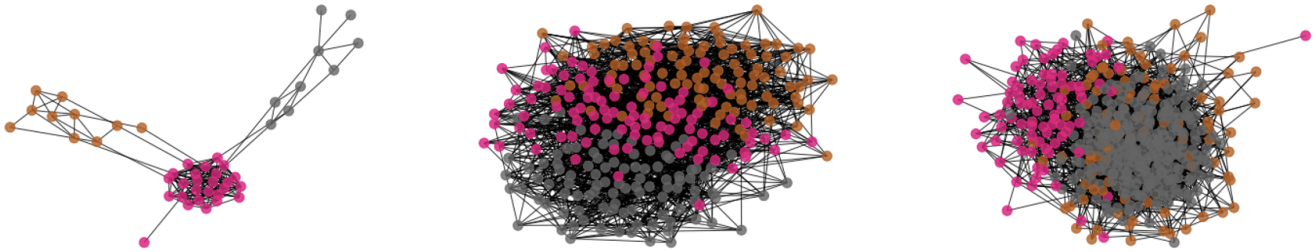
TABLE 1: Specifications for Generated Graphs

These three test cases range from simple to complex from a community detection perspective. Complexity is added by making the graphs bigger, and the probabilities more opaque. The generated graphs can visually be seen in Figure 5a. Here, colors correspond to the true community assignments for the nodes.

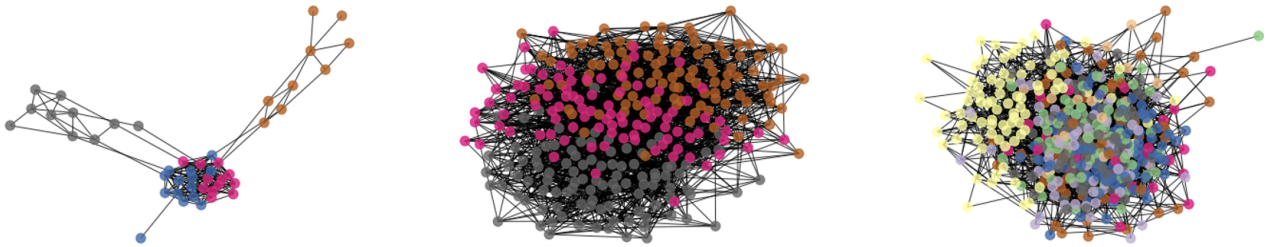
4. To intuitively understand this metric, consider the case where X and Y are independent. Then $p_{X,Y}(x,y) = p_X(x)p_Y(y)$, and thus $I(X;Y) = 0$. So observing X reveals no information about Y as it is an independent process.



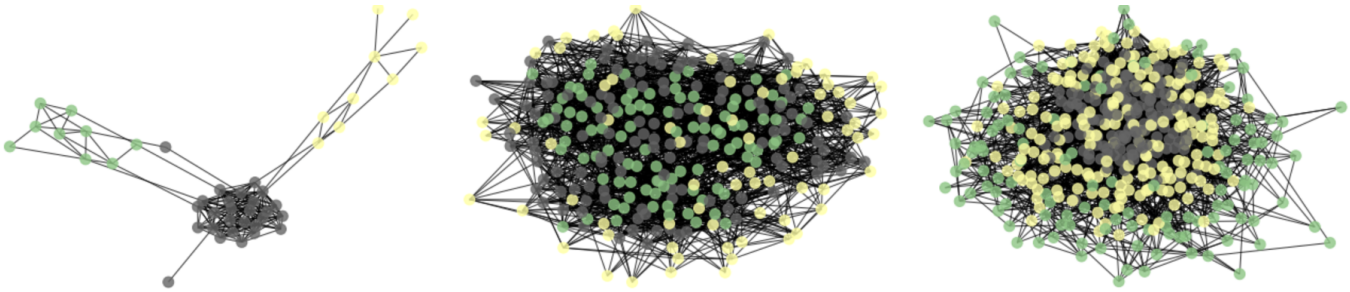
(a) Original Graphs (Ground Truth)



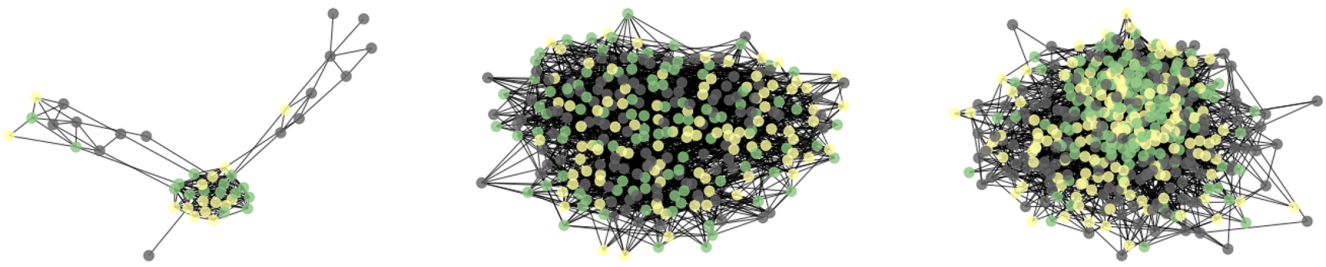
(b) Labels from Spectral Clustering



(c) Labels from Louvain's Algorithm



(d) Labels from Graph Encoder



(e) Labels from MinCutPool

Fig. 5: A Visualisation of the Performance of our Methods on Synthetic Graphs for the Simple, Moderate and Complex Cases (moving left to right) as defined in Table 1.

5 Results

We report average NMI scores from 50 runs for all five algorithms explained in Section 3. The results are summarised in Table 2.

	Simple	Moderate	Complex
Spectral Clustering	**0.915**	**0.983**	**0.686**
Louvain's Algorithm	0.773	0.872	0.286
Graph Encoder	0.714	0.040	0.218
MinCutPool	0.410	0.018	0.111
SDCN	0.883	0.793	0.492

TABLE 2: NMI for Synthetically Generated Graphs

A visualisation of the performance can be seen in Figure 5. As seen by the NMI scores, the best performance is attained by Spectral Clustering across all three levels of complexity, followed closely by Louvain's Algorithm. Since Louvain's Algorithm only terminates when no more greedy merging is possible, it does not take in the number of desired clusters apriori, and in our experiments it yielded four cluster partitions frequently even though the true graph had only three clusters. It's interesting to note that both Spectral Clustering and Louvain's Algorithm achieved superior performance by NMI on the moderate case as compared to the simple case. One of the reasons for this may be that while the moderate case was more difficult than the simple case in the sense that the probabilities were more opaque (there was a smaller discrepancy between p and q), it was 'easier' in the sense that it had all 3 communities balanced whereas the simple case had 2 small and 1 larger community. This suggests an interesting insight for these two methods: they are relatively robust to opacity of the graph structure but are sensitive to imbalanced community size. This motivates an interesting investigation for further study of these methods, perhaps by adversarially constructing graphs of unbalanced cluster sizes and studying the effect on performance.

In contrast to the two non-deep learning approaches, the three deep learning approaches struggle to achieve high NMI scores, especially on the moderate and complex cases. Further note that for the Graph Encoder, results were highly unstable as the NMI score obtained varied between 0.6 and 1.0 based on the run. Using K-means++ to intialize the last step ameliorated this to an extent. We conjecture several potential causes for why the deep learning approaches showed inferior performance despite their success in other use cases in literature[5].

First, note that the Graph Encoder begins with a graph as well as a similarity matrix that are used as initial autoencoder inputs. In the original paper [Tian et al., 2014], the authors used non-graph features for computing the similarity matrix via cosine similarity. For purely graphical datasets with no additional features, it was not specified exactly how the similarity matrix would be computed. We used cosine similarity on binary vectors from the adjacency matrix to compute this score for our experiments, but perhaps a better approach may be desired in future attempts to get superior performance. The MinCutPool also suffers from a similar problem; most of the experiments in the original paper [Bianchi et al., 2019] relied in a graph structure as well as a rich set non-graph features for individual nodes that were fed as inputs into the GCN to be embedded and eventually clustered. Since our use case was purely graphical, we simply imputed graph metrics such as node PageRank and clustering coefficient in place of these features. We speculate that the success of these methods was largely a product of the rich feature set used in these methods, and the graphical structure simply provided incremental performance enhancements. Since our experimental setup had no such information from individual features (we only had a given graph structure), the methods did not achieve impressive clustering performance. Of course, this remains a speculation currently, and can only be verified by another round of careful experimentation in a randomized trial fashion[6].

Finally, the SDCN turned out to be the relatively reliable in our tests and while it did not manage to beat the Spectral clustering in terms of the NMI, it on average performs better than the Louvain's algorithm, however, the number of clusters must be pre-specified unlike in Louvain. We believe that the performance gain partially comes from the use of the pre-training of the autoencoder, which is then only fine-tuned to the clustering problem by the dual self-supervised module.

6 Conclusion

In our work we have explored the graph clustering algorithms, the ones that do not rely on deep learning methods as well as the ones that use state-of-the-art deep learning concepts such as autoencoders and graph convolutional networks. We provided a detailed theoretical review of all methods that we used in our work, moreover, we have implemented all defined methods and compared their performance on a set of three different synthetic scenarios generated using the stochastic block model. We have demonstrated that the Spectral Clustering was clearly the best out of all the algorithms with an average NMI over 90%, the second best was the neural architecture based SDCN. Moreover, we have discussed in detail why the other two deep learning methods that we employed may not have done too well. We also note that our work was more concerned about the precision of the clustering on specific, but relatively small cases. However, due to the limited scope of the work, we did not evaluate performance on very large graphs, where runtime may be of essence and a neural approach such as SDCN may provide comparable performance with a much lower runtime.

5. Albeit, not always for community detection tasks

6. An experiment where we would run the methods with and without graph structure keeping all the features constant.

REFERENCES

[Arthur and Vassilvitskii, 2006] Arthur, D. and Vassilvitskii, S. (2006). k-means++: The advantages of careful seeding. Technical report, Stanford.

[Bianchi et al., 2019] Bianchi, F. M., Grattarola, D., and Alippi, C. (2019). Mincut pooling in graph neural networks. *CoRR*, abs/1907.00481.

[Bianchi et al., 2020] Bianchi, F. M., Grattarola, D., and Alippi, C. (2020). Spectral clustering with graph neural networks for graph pooling. In *International Conference on Machine Learning*, pages 874–883. PMLR.

[Blondel et al., 2008] Blondel, V. D., Guillaume, J.-L., Lambiotte, R., and Lefebvre, E. (2008). Fast unfolding of communities in large networks. *Journal of statistical mechanics: theory and experiment*, 2008(10):P10008.

[Bo et al., 2020a] Bo, D., Wang, X., Shi, C., Zhu, M., Lu, E., and Cui, P. (2020a). Structural deep clustering network. In *Proceedings of The Web Conference 2020*, pages 1400–1410.

[Bo et al., 2020b] Bo, D., Wang, X., Shi, C., Zhu, M., Lu, E., and Cui, P. (2020b). Structural deep clustering network. In *Proceedings of The Web Conference 2020*, pages 1400–1410.

[Bottou, 1999] Bottou, L. (1999). *On-Line Learning and Stochastic Approximations*, page 9–42. Cambridge University Press, USA.

[Bottou and Bengio, 1994] Bottou, L. and Bengio, Y. (1994). Convergence properties of the k-means algorithms. *Advances in neural information processing systems*, 7.

[Bresar et al., 2011] Bresar, B., Kardos, F., Katrenic, J., and Semanis in, G. (2011). Minimum k-path vertex cover. *Discrete Applied Mathematics*, 159(12):1189–1195.

[Cortez et al., 2009] Cortez, P., Cerdeira, A., Almeida, F., Matos, T., and Reis, J. (2009). Modeling wine preferences by data mining from physicochemical properties. *Decision support systems*, 47(4):547–553.

[Danon et al., 2005] Danon, L., Diaz-Guilera, A., Duch, J., and Arenas, A. (2005). Comparing community structure identification. *Journal of statistical mechanics: Theory and experiment*, 2005(09):P09008.

[De Meo et al., 2011] De Meo, P., Ferrara, E., Fiumara, G., and Provetti, A. (2011). Generalized louvain method for community detection in large networks. In *2011 11th international conference on intelligent systems design and applications*, pages 88–93. IEEE.

[Dhillon et al., 2004] Dhillon, I. S., Guan, Y., and Kulis, B. (2004). Kernel k-means: Spectral clustering and normalized cuts. In *Proceedings of the Tenth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '04, page 551–556, New York, NY, USA. Association for Computing Machinery.

[Fix and Hodges, 1989] Fix, E. and Hodges, J. L. (1989). Discriminatory analysis. nonparametric discrimination: Consistency properties. *International Statistical Review/Revue Internationale de Statistique*, 57(3):238–247.

[Gach and Hao, 2013] Gach, O. and Hao, J.-K. (2013). Improving the louvain algorithm for community detection with modularity maximization. In *International Conference on Artificial Evolution (Evolution Artificielle)*, pages 145–156. Springer.

[Gardner and Dorling, 1998] Gardner, M. W. and Dorling, S. (1998). Artificial neural networks (the multilayer perceptron)—a review of applications in the atmospheric sciences. *Atmospheric environment*, 32(14-15):2627–2636.

[Holland et al., 1983] Holland, P. W., Laskey, K. B., and Leinhardt, S. (1983). Stochastic blockmodels: First steps. *Social Networks*, 5(2):109–137.

[Huttenhower et al., 2007] Huttenhower, C., Flamholz, A. I., Landis, J. N., Sahi, S., Myers, C. L., Olszewski, K. L., Hibbs, M. A., Siemers, N. O., Troyanskaya, O. G., and Coller, H. A. (2007). Nearest neighbor networks: clustering expression data based on gene neighborhoods. *BMC bioinformatics*, 8(1):1–13.

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

[Kipf and Welling, 2017] Kipf, T. N. and Welling, M. (2017). Semi-supervised classification with graph convolutional networks. 2017. *ArXiv abs/1609.02907*.

[Lahitani et al., 2016] Lahitani, A. R., Permanasari, A. E., and Setiawan, N. A. (2016). Cosine similarity to determine similarity measure: Study case in online essay assessment. In *2016 4th International Conference on Cyber and IT Service Management*, pages 1–6. IEEE.

[Liu et al., 2018] Liu, G., Bao, H., and Han, B. (2018). A stacked autoencoder-based deep neural network for achieving gearbox fault diagnosis. *Mathematical Problems in Engineering*, 2018.

[Lloyd, 1982] Lloyd, S. (1982). Least squares quantization in pcm. *IEEE transactions on information theory*, 28(2):129–137.

[MacQueen et al., 1967] MacQueen, J. et al. (1967). Some methods for classification and analysis of multivariate observations. In *Proceedings of the fifth Berkeley symposium on mathematical statistics and probability*, volume 1, pages 281–297. Oakland, CA, USA.

[Malliaros and Vazirgiannis, 2013] Malliaros, F. D. and Vazirgiannis, M. (2013). Clustering and community detection in directed networks: A survey. *Physics reports*, 533(4):95–142.

[Nascimento and De Carvalho, 2011] Nascimento, M. C. and De Carvalho, A. C. (2011). Spectral methods for graph clustering–a survey. *European Journal of Operational Research*, 211(2):221–231.

[Newman and Girvan, 2004] Newman, M. E. and Girvan, M. (2004). Finding and evaluating community structure in networks. *Physical review E*, 69(2):026113.

[Ng et al., 2001] Ng, A., Jordan, M., and Weiss, Y. (2001). On spectral clustering: Analysis and an algorithm. *Advances in neural information processing systems*, 14.

[Nwankpa et al., 2018] Nwankpa, C., Ijomah, W., Gachagan, A., and Marshall, S. (2018). Activation functions: Comparison of trends in practice and research for deep learning. *arXiv preprint arXiv:1811.03378*.

[Shi and Malik,] Shi, J. and Malik, J. Normalized cuts and image segmentation. *Proceedings of IEEE Computer Society Conference on Computer Vision and Pattern Recognition*.

[Tian et al., 2014] Tian, F., Gao, B., Cui, Q., Chen, E., and Liu, T.-Y. (2014). Learning deep representations for graph clustering. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 28.

[Tsitsulin et al., 2020] Tsitsulin, A., Palowitch, J., Perozzi, B., and Müller, E. (2020). Graph clustering with graph neural networks. *arXiv preprint arXiv:2006.16904*.

[Ushioda, 1996] Ushioda, A. (1996). Hierarchical clustering of words and application to nlp tasks. In *Fourth Workshop on Very Large Corpora*.

[Van der Maaten and Hinton, 2008] Van der Maaten, L. and Hinton, G. (2008). Visualizing data using t-sne. *Journal of machine learning research*, 9(11).

[Vincent et al., 2010] Vincent, P., Larochelle, H., Lajoie, I., Bengio, Y., Manzagol, P.-A., and Bottou, L. (2010). Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion. *Journal of machine learning research*, 11(12).

[Von Luxburg, 2007] Von Luxburg, U. (2007). A tutorial on spectral clustering. *Statistics and computing*, 17(4):395–416.

[White and Smyth, 2005] White, S. and Smyth, P. (2005). A spectral clustering approach to finding communities in graphs. In *Proceedings of the 2005 SIAM international conference on data mining*, pages 274–285. SIAM.

[Xie et al., 2016] Xie, J., Girshick, R., and Farhadi, A. (2016). Unsupervised deep embedding for clustering analysis. In *International conference on machine learning*, pages 478–487. PMLR.

[Yang et al., 2017] Yang, B., Fu, X., Sidiropoulos, N. D., and Hong, M. (2017). Towards k-means-friendly spaces: Simultaneous deep learning and clustering. In *international conference on machine learning*, pages 3861–3870. PMLR.

[Zhou et al., 2020] Zhou, J., Cui, G., Hu, S., Zhang, Z., Yang, C., Liu, Z., Wang, L., Li, C., and Sun, M. (2020). Graph neural networkpectral: A review of methods and applications. *AI Open*, 1:57–81.