UNIVERSITY OF CAMBRIDGE
Department of Computer Science and Technology

Research project report title page

Candidate **2115I**

"Community detection using structure-derived features"

Submitted in partial fulfilment of the requirements for the Master of Philosophy in Advanced Computer Science

Abstract

Community detection using structure-derived features.

In this work we explore the intersection between spectral and spatial methods for community detection, focusing on graph data with hierarchical properties. We design an algorithm to recursively partition an input graph into two using the Fiedler vector to inform the choice of cut. Spectral analysis of the input and its resultant subgraphs produces eigenvectors which are subsequently used as node features for graph neural networks (GNNs).

The models are trained on four classes of synthetic data with labelled communities, utilising an efficient approach for permuting community predictions based on solving a bipartite graph matching problem. Included among the synthetic classes are stochastic block models and hierarchical networks, and we analyse their markedly different characteristics. We assess the importance of our structure-derived features, as well as node-local features such as node degree. In tandem we explore variations on the Laplacian, methods for choosing the cut, and the addition of 'virtual nodes' that enhance connectivity of the (possibly disconnected) subgraphs. Also investigated are two alternative loss functions that can be used to train models in the absence of labelled data, as well as two entropy-based regularisation functions. Additionally, we reframe community detection as an edge prediction task, and leverage this to implement a model that directly targets information bottlenecks in GNNs. This is evaluated on an enzyme classification task, though we discover no improvement over the baseline model.

The community detection models are evaluated on the Cora and CiteSeer citation networks, and we observe that the additional features do not contribute sufficient new information to improve over the baseline. However, we discover that even when we drop the node attributes, we can recover most of the underlying community structure. This is an important finding that shows the utility of our approach in domains where attribute data may be corrupted.

Word count: 14,576

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Community structure arises in many networks, where nodes in the same community share similar properties. Well-known examples include social networks, where users in the same community may be more likely to share a connection, and citation networks, with papers in the same academic field more likely to cite one another. The task of community detection, or clustering, is to recover the underlying communities of a graph from the node features and adjacency information. Recent advances in the field employ graph neural networks (GNNs) for this purpose, which offer significant advantages over other machine learning (ML) approaches. Specifically, models such as k-means clustering and Multi-Layer Perceptron (MLP) networks are restricted to making clustering decisions from the node attributes only, and therefore classify independently from one another. In contrast, GNNs incorporate adjacency information via message-passing to derive a representation about both a node *and* its local k-hop neighbourhood.

A clear limitation in focusing on a local neighbourhood, however, is the inability to capture global information about the graph. A key challenge of working with generic graph-structured data is the lack of any natural sense of direction that is present in data such as images or grids. This property makes it challenging to infer the topographical location of a node within a graph, and consequently to identify clusters determined by 'location'. In graphs with featureless nodes, any clustering has to be carried out using only the graph structure, without relying on features, e.g. as bag-of-words representations of papers in a citation network.

Two techniques to derive structural features are to use node-local information such as node degree, or at the global scale, to utilise spectral graph theory. The latter focuses on the insight gained when we decompose an adjacency matrix into its eigenvalues and eigenvectors, and this work makes extensive use of spectral information. In particular, we seek to answer three questions:

1. To what extent can spectral decomposition aid community detection in the absence of node attributes?
2. Can changes to decomposition techniques be made to suit particular properties of graphs?
3. How do spectrally-derived features compare to locally-derived features?

We begin by establishing the context for this work in Chapter 2. We then set out the requisite background information to help understand the notation and motivation for our methodology, in Chapters 3 and 4. In Chapter 5 we elaborate on the datasets used in this project – we focus on four synthetic families of network data. Here we outline the importance of hierarchical networks and their particular qualities. We carry out an extensive evaluation of the synthetic data in Chapter 6, and demonstrate the results of our experiments with the well-known Cora citation network. The implications of our work are discussed in Chapter 7, before Chapter 8 summarises our findings.

Chapter 2

Related work

Community detection is a long-standing problem in network theory and there exists a variety of algorithmic approaches. Blondel *et al.* [1] find high-quality communities in large graphs by optimising for Newman's measure of graph modularity, which maximises the within-community edge density compared to the between-community [2]. Blondel *et al.* show an efficient method for calculating the *change* in modularity that occurs when an isolated node u is moved from community C_i to C_j. Communities are then built bottom-up, starting by assigning each node to its own community then making multiple passes over the graph, moving – or not moving – each node to a neighbouring community such that modularity is maximised. A notable drawback of this approach is the assumption that communities are inherently modular. As Leskovec *et al.* observe, large communities can tend to "blend into" the rest of the graph, leading to poor modularity [3]. They instead propose a measure based on conductivity to assess community quality. Chen, Nguyen and Szymanski also find modularity can unjustly favour large communities over small, as well as the converse depending on the circumstance, and evaluate measures to rectify this [4]. This kind of community detection can be used to illuminate similarities between nodes, such as identifying functional subunits in protein interactions [5] or revealing users with shared interests in social networks [6].

Alternatively, communities may facilitate graph pooling, where the underlying structure of a graph can be simplified by coalescing groups of nodes into a new, lower-resolution representation, that may be used for subsequent classification tasks. These approaches can be divided broadly into either global or hierarchical pooling. The former category typically involves using neural networks to process nodes to form a task-specific representation, then applying a readout operation that aggregates information from all nodes into a single vector. This vector then acts as input to a classification network, an approach succinctly captured by Gilmer's *et al.* Message Passing Neural Networks [7]. For the readout function, Gilmer *et al.* use either a pair of neural networks, or the set2set operator proposed by Vinyals *et al.* [8]. A related work is Zhang's *et al.* SortPool, which learns a useful *ordering* of node; this can then be passed to a 1-D convolutional layer for further processing [9]. Note that the global pooling approach does not generate useful community representations. In contrast, hierarchical pooling employs multiple pooling steps, iteratively unearthing increasingly coarser representations of the input graph. Graph classification can be implemented by setting the desired number of nodes in the output layer to 1. The approach shared by DiffPool (Ying *et al.* [10]), EigenPool (Ma *et al.* [11]) and LaPool (Noutahi *et al.* [12]) is to learn a differentiable function that maps each of the N_L nodes in a given layer to a vector of size N_{L+1}. This is represented by the soft-assignment matrix $S \in \mathbb{R}^{N_L \times N_{L+1}}$. Crucially, a new structural representation of the graph is produced at each layer by calculating $A_{L+1} = S_L^T A_L S_L$, where A_L is the adjacency matrix at layer L; this can be used to inspect the task-specific communities.

Another approach to pooling is to make use of repeating motifs. Jin *et al.* use molecular motifs with an encoder-decoder architecture that learns a hierarchical representation of molecular graphs, which can then be used to generate new molecules for drug discovery [13]. However, their model does not learn to identify the motifs itself, instead relying on a separate domain-specific motif extraction algorithm. In fact, GNNs find identifying subgraphs challenging, and as Chen *et al.* demonstrate, the entire class of message-passing GNN architectures are incapable of counting *induced subgraphs* of 3 or more nodes [14]; that is, for a graph $G = (V, E)$, any induced subgraph $G^S = (V^S, E^S)$ includes in E^S all edges in E with both endpoints in V^S. Xu *et al.* also show that classical GNN architectures are at most as expressive as the WL test for graph isomorphism [15]. Additionally, they develop the Graph Isomorphism Network that is at least as expressive as the Weisfeiler-Lehmen (WL) test.

Upper expressivity bounds aside, GNNs are nonetheless a powerful tool for community detection. Bruna approaches community detection using multiscale GNNs [16]. These aggregate information from within a k-hop neighbourhood within a single layer, as opposed to stacking multiple layers to incorporate the same size neighbourhood. He then adds residual connections [17] to a degree-scaled feature matrix, and a globally aggregated feature matrix. This allows the model to perform power iteration, an efficient algorithm for finding eigenvectors [18], and hence the model is capable of learning spectral information. Bruna *et al.* compare spectral clustering to naïve agglomerative methods and show that spectral methods are indeed better at capturing features of the whole-graph structure [19]. In particular, spectral methods resulted in more “spatially delocalised” clusters. However, spectral analysis has its own shortcomings, as highlighted by Guattery and Miller who show that there exist certain families of graph for which spectral analysis cannot produce an optimal cut [20]. Various techniques exist to overcome this, including the use of ‘spectral maximiser’ graphs shown by Koutis and Le [21], and other approaches that directly target the Laplacian matrix which we discuss in detail in Section 4.1.1.

The multiscale approach is also adoped by Ruiz *et al.* to predict the effects of drug treatments, by using a diffusion profile of interactions between drugs, proteins biological functions, and diseases [22]. They observe that in many treatments, drugs interact with diseases not physically at the molecular level, but instead through a complex chain of hierarchical interactions. Modelling for this, they are able to significantly outperform the molecular-scale approach. Importantly, this is achieved not through increased algorithmic efficiency, but through a paradigm shift to the use of multiple resolutions within the interaction network. Similarly, clustering at different resolutions is also successfully employed by Shavit *et al.* to predict how chromosomes fold, based on contact maps between chromatin fragments [23].

Chapter 3

Spectral graph theory

One application of graph clustering techniques is in image segmentation tasks, as put forth by Shi and Malik [24]. They represent an image as a graph $G = (V, E)$ with pixels comprising the set of vertices V and edges E weighted by similarity of node features, such as colour. Segmentation can be performed using a top-down approach, by cutting the graph into two, then repeating the process on the resulting partitions, until some termination criteria are reached. The challenge is to find an optimal cut that minimises the weight of edges between two partitions A and B normally given by

$$cut(A,B) = \sum_{i \in A,\ j \in B} w_{i,j} \quad ,$$

where $A, B \in V \mid A \cup B = V, A \cap B = \varnothing$. However a clear disadvantage to $cut(A, B)$ is that no concern is given to trying to ensure evenly-sized partitions. For example, in an unweighted connected graph, a node with only a single edge can be partitioned alone as an effective solution. Therefore a preferred measure is to use the normalized cut

$$Ncut(A,B) = \frac{cut(A,B)}{assoc(A,V)} + \frac{cut(B,A)}{assoc(B,V)} \quad ,$$

with $assoc(A,V)$ being functionally equivalent to $cut(A,V)$ but dropping the condition that $A \cap V = \varnothing$. As *cut* is symmetric, $cut(B,A) = cut(A,B)$. The denominator terms serve to maximise the sum of node degrees for each node in A and B. We use the notation $d_i = \sum_{j \in V} w_{i,j}$ to denote node degree, with the corresponding degree matrix D defined such that $D_{i,j} = d_i \ \forall \ i = j$ and $D_{i,j} = 0$ otherwise.

Shi and Malek express the cut using a $|V|$-dimensional vector $\mathbf{x}$ defined such that $\mathbf{x}_i = 1$ if $i \in A$ and $\mathbf{x}_i =$ -1 if $i \in B$. Indicator vectors $\frac{1+\mathbf{x}}{2}$ and $\frac{1-\mathbf{x}}{2}$ are used to describe A and B, where 1 indicates membership, and 0 absence. An expression for $cut(A,B)$ can be found by starting with the edge weights for all nodes in A, i.e. $\sum_{i \in A} w_{i,j} = d_i \ \forall \ i \in A$. This can be alternately expressed as $(1+\mathbf{x})^T D(1+\mathbf{x})$, where W is the weight matrix for G. We can similarly express the edges contained entirely within A as $(1+\mathbf{x})^T W(1+\mathbf{x})$. As the cross-partition edges can be inferred as the difference of all the edges contained entirely within A from the edges with at least one node in A, it can be seen that

$$4[Ncut(A,B)] = \frac{(1+\mathbf{x})^T(D-W)(1+\mathbf{x})}{d_A} + \frac{(1-\mathbf{x})^T(D-W)(1-\mathbf{x})}{d_B} \quad ,$$

where d_A, d_B are the respective sums of node degrees in partitions A and B. Shi and Malek show, (full derivation omitted), that this expression is equivalent to

$$\frac{[(1+\mathbf{x}) - b\,(1-\mathbf{x})]^T(D-W)[(1+\mathbf{x}) - b\,(1-\mathbf{x})]}{b\,\mathbf{1}^T D\mathbf{1}} \quad ,$$

where $b = \frac{d_A}{d_B}$ is the ratio of node degrees between A and B, (so $d_A = b\,d_B$), and $\mathbf{1}$ is the all-ones column vector of size $|V|$. By substituting $\mathbf{y} =\ (1+\mathbf{x}) - b\,(1-\mathbf{x})$, it can be seen that

$$b\,\mathbf{1}^T D\mathbf{1} = b\,(d_A + d_B) = b\,d_A + d_A = b^2\,d_B + d_A = \mathbf{y}^T D\mathbf{y} \quad ,$$

and hence minimising $Ncut(A, B) = Ncut(\mathbf{x})$ with respect to $\mathbf{x}$ is equivalent to minimising

$$\frac{\mathbf{y}^T(D - W)\mathbf{y}}{\mathbf{y}^T D\mathbf{y}} \tag{3.1}$$

with respect to $\mathbf{y}$. This is subject to two constraints:

1. The vector $\mathbf{y}$ is composed solely from elements $\in \{1, -b\}$.
2. By design, $d_A - b\, d_B = 0$, so consequently $\mathbf{y}^T D\mathbf{1} = 0$.

The discrete nature of condition 1 is onerous, making the optimization problem NP-hard, so we relax this and look instead for a real-valued vector, leaving aside the problem of assigning nodes to partitions for now. Setting up a generalised eigenvalue problem with the symmetric matrices D and $(D-W)$

$$(D - W)\mathbf{y} = \lambda D\mathbf{y} \tag{3.2}$$

makes it clear that minimising 3.1 with respect to $\mathbf{y}$ is equivalent to minimising λ with respect to $\mathbf{y}$. Substituting $\mathbf{y} = D^{\frac{1}{2}}\,\mathbf{z}$ transforms 3.2 to a standard eigensystem

$$D^{-\frac{1}{2}}(D - W)D^{-\frac{1}{2}}\mathbf{z} = \lambda\mathbf{z}\ .$$

If we suppose $\lambda_0 = 0$, it can be seen that $\mathbf{z}_0 = D^{\frac{1}{2}}\,\mathbf{1}$ is a valid eigenvector: by construction, the sums of each row of W are equal to the diagonal of D, and hence $(D\ -\ W)\,\mathbf{1}\ =\ \mathbf{0}$. Note that the eigenvector $\mathbf{z}_0$ corresponds to $\mathbf{y}_0 = \mathbf{1}$, i.e. the assignation of all nodes to partition A, for which $Ncut(A, B)$ is undefined but that nevertheless clearly satisfies condition 2.

The matrix $(D - W)$ is positive semi-definite [25], and hence its eigenvalues are all orthogonal and non-negative – with λ_0 being the smallest. We can hence derive an expression for the second-smallest eigenvector $\mathbf{y}_1$ using the fact that $\mathbf{z}_1^T\mathbf{z}_0 = 0$; this is equivalent to $\mathbf{y}_1^T D\mathbf{1}$ and so also satisfies condition 2.

Choosing the next-smallest eigenvector can be carried out using the Rayleigh Quotient [26, p. 453]. Given eigenvectors $\mathbf{y}_0, \dots, \mathbf{y}_{k-1}$ of a matrix A, then

1. $\mathbf{y}_0, \dots, \mathbf{y}_{k-1}$ will be orthogonal if A is symmetric;
2. The next-smallest eigenvalue can be found as
$$\lambda_k = \min_{\mathbf{y} \perp \mathbf{y}_0, \cdots, \mathbf{y}_{k-1}} \frac{\mathbf{y}^T A \mathbf{y}}{\mathbf{y}^T \mathbf{y}} \quad ;$$
3. The corresponding eigenvector is found as
$$\mathbf{y}_k = \operatorname*{arg\,min}_{\mathbf{y} \perp \mathbf{y}_0, \cdots, \mathbf{y}_{k-1}} \frac{\mathbf{y}^T A \mathbf{y}}{\mathbf{y}^T \mathbf{y}} \quad . \tag{3.3}$$

Comparing 3.3 with 3.1, then, demonstrates that the second-smallest eigenvector of $D^{-\frac{1}{2}}(D - W)D^{-\frac{1}{2}}$ minimises the normalized cut. This $\mathbf{y}_1$ vector effectively places an ordering over all nodes in G, and candidate solutions for the discrete problem can be drawn, for instance, with a vector such as $\mathbf{y}^*$ defined such that $\mathbf{y}^*_i = 1$ if $\mathbf{y}_{1_i} > 0$, or $-b$ otherwise.

The importance of the $(D - W)$ matrix is not coincidental: it is known as the Laplacian matrix of any graph G, and its eigenvectors provide substantial insight into the structure of G. The matrix $D^{-\frac{1}{2}}(D - W)D^{-\frac{1}{2}}$ is also known as the symmetric normalized Laplacian L^{sym}; it scales every element $L^{sym}_{i,j}$ by dividing by $\sqrt{d_i d_j}$, with elements $L^{sym}_{i,i}$ on the main diagonal consequently equal to 1 wherever $d_i \neq 0$.

Shi and Malek discuss how subsequent eigenvectors $\mathbf{y}_{>1}$ can be used to recursively find the normalized cuts of the partitions A, B and so forth, though in practice this procedure cannot be continued indefinitely due to approximation errors [24]. The second-smallest eigenvalue λ_1 in particular is referred to by Fiedler as the algebraic connectivity of a graph $a(G)$, has the important property that G is connected iff $a(G) > 0$ [27]; its corresponding eigenvector is sometimes known as the Fiedler vector. Moreover, the number of connected components in G is precisely equal to the multiplicity of $\lambda_0 = 0$ [28].

Chapter 4

Methodology

4.1 Feature generation

4.1.1 Spectral decomposition

Prior to model training, spectral analysis is performed on all input graphs, and this forms the basis for node features. Principally, the k-smallest eigenvalues, with their corresponding eigenvalues, are derived from the Laplacian matrix L of an input graph G. We restrict our analysis to graphs consisting of a single connected component, and hence λ_1 will be non-zero, with $\mathbf{z}_1$ being the Fiedler vector that approximates the solution to the normalised cut. As discussed in Chapter 3, the $\mathbf{z}_1$ vector represents the 'strongest' sense of direction in any graph. For example, [dgn] find that for grid graphs, $\mathbf{z}_1$ increases monotonically in the longest direction. This can be easily verified using the statements made in Chapter 3, when we note that the optimal solution for the normalised cut of a grid graph would be the midway point along its longest direction. As Shi and Malik note, the $\mathbf{z}_2$ vector indicates the optimal normalised cut among nodes partitioned by $\mathbf{z}_1$; clearly, this corresponds to the midway point along the shortest dimension of the grid, and thus the pair taken together partition the graph into four quadrants.

Excluding $\mathbf{z}_0$ then, there are $|V| - 1$ eigenvectors that could be derived from

L and used as node features. However in practice the accumulated approximation errors incurred in calculating subsequent vectors mean often only the first few provide useful information for clustering, as we partially show in the experiments of Section 6.1.4. These first eigenvectors can discriminate effectively between nodes that are far away from one another, and hence used to perform low-resolution clustering, of the kind that is not achievable using local features such as node degree, or message passing and aggregation mechanisms where information becomes saturated as it is communicated over longer distances.

If we want to perform clustering at smaller scales, this low-frequency information is less useful. One approach would be to perform large-scale clustering to partition the input G_0 into subgraphs $G_{1,0} \ldots G_{1,c_0}$, then repeat the spectral analysis to derive the first few eigenvectors of each of the subgraphs. Clearly, this process can be repeated recursively as desired, partitioning $G_{1,0}$ into subgraphs $G_{1,0,0} \ldots G_{1,0,c_1}$ and so forth. This is analogous to a deriving a 'postcode system' for the nodes of the graph, and allows clustering mechanisms to use information at various scales.

An obvious impediment to this approach is that having to learn to first perform large-clustering, then incrementally stacking spectral analyses and increasingly finer clustering operations makes for a very deep system, and useful learning signals are unlikely to propagate sufficiently. Instead we perform this recursive decomposition offline, as a preprocessing step. To achieve this, we need an algorithmic way of partitioning the graph. Clearly, having already calculated the first k eigenvectors, an obvious approach is to use $\mathbf{z}_1$, approximating as it does the normalised cut. However, being an approximation, $\mathbf{z}_1$ assigns values to each node only on a continuous scale, when in practice we require the discrete assignment vector $\mathbf{y}$ discussed in Chapter 3, which was composed of elements $\in \{1, -b\}$ that indicate to which partition each node should belong.

We therefore require a method to convert $\mathbf{z}_1$ into an assignment vector. In Section 6.1.3 we compare three approaches: '*median*', '*sign*' and '*ncut*'. For the 'median' method, we take the median value $med(\mathbf{z}_1)$ and create two partitions A and B, with node $i \in A$ if $\mathbf{z}_{1,i} < med(\mathbf{z}_1)$ and node $j \in B$ if $\mathbf{z}_{1,j} \geqslant med(\mathbf{z}_1)$. The 'sign' method assigns nodes based on the sign of their respective elements in $\mathbf{z}_1$, i.e. $i \in A$ if $\mathbf{z}_{1,i} < 0$ and vice-versa. Lastly, the 'ncut' assignment searches for an approximate solution to the normalised cut. This is achieved by sorting the nodes according to their values in $\mathbf{z}_1$, then calculating $ncut(A, B)$ for multiple split points – rather than only $med(\mathbf{z}_1)$ or 0 for 'median' and 'sign' – and using whichever partition most minimises $ncut(A, B)$. For larger graphs, Shi and Malik find that setting a limit on the number of split points and evenly distributing these across $\mathbf{z}_1$ returns a sufficiently good solution. We use a similar approach, but normally distribute the split points around the mean of $\mathbf{z}_1$, to increase the likelihood of finding a more optimal normalised cut.

It should be noted that the resulting partitions may not be connected. For example, the only way to partition a star-like graph into two subgraphs while ensuring both subgraphs are connected is to partition a single node – not the centre node – leaving the remaining graph intact. We can still perform spectral analysis on a disconnected graph, so this is not impractically problematic. However, in Section 6.1.2 we carry out experiments that maintain the connectivity of resulting subgraphs by use of 'virtual nodes'. Two varieties are explored, with either single or multiple virtual nodes. For single virtual nodes, after a graph has been split into partitions A and B, all edges between partitions are instead connected to a new node. That is, any edge $(i, j) \in \mathrm{E}$ s.t. $i \in A$, $j \in B$ is replaced by the edges (i, v_1) and (j, v_2), where $v_1 \in A$ and $v_2 \in B$ are the new virtual nodes added to each subgraph. However, this approach often results in the single virtual node having unusually high degree, which can distort the degree distribution of the subgraph. Consider that nodes with cross-partition edges are more likely to be found on the periphery; by connecting all such nodes to the same virtual node, this significantly reduces their eccentricity, shifting the centre of the graph towards them. To address this, a variation is used that uses a different virtual node for each other partition of the original graph G with an edge to the current subgraph. We explore whether to weight the edges to the virtual node. As a single node in A may have multiple cross-partition edges, we choose whether to set its edge weight to the virtual node equal to the sum of weights of all such edges, or simply set this to 1 for the unweighted variation.

Variants of the Laplacian

As we have seen in Chapter 3, the $\mathbf{z}_1$ vector minimises $Ncut$. Alternative forms of the Laplacian exist which provide a different outlook on the structure of G. The $\mathbf{z}_1$ vector of the unnormalised Laplacian $L = D - W$ can be shown to minimise the $RatioCut$, defined as

$$RatioCut(A, B) = \frac{cut(A, B)}{|A|} + \frac{cut(B, A)}{|B|} .$$

Hamilton provides an outline of this in [29, pp. 24–26].

The random-walk Laplacian is another variant, defined as

$$L^{RW} = D^{-1}L .$$

In a random graph walk, a stochastic process moves randomly between nodes, with the probability of traversing from node u to node v given by $\frac{1}{d_u}W_{u,v}$. This means we can interpret L^{RW} as a transition matrix giving the probabilities of any given traversal. Further, von Luxburg notes that minimising $Ncut(A, B)$ can be interpreted as finding a cut where a random walk only rarely transitions between A and B [30].

Amini *et al.* find that spectral clustering methods using L^{sym} performs worse for community detection in sparser graphs with a low average node degree $\mathbb{E}(d)$, finding this was most noticeable once $\mathbb{E}(d) < 5$ [31]. They evaluate on graphs that may contain multiple components, and suggest the cause of poor spectral representations is a result of eigenvectors not discriminating well within disconnected components. To counter this, they propose 'spectral clustering with perturbations', whereby low-weight edges are added in order to connect components. They add to W a constant matrix of small values, scaled by $\mathbb{E}(d)$, then carry out standard spectral clustering using k-means. This was found to be particularly successful for graphs with weighted edges.

Motivated by a similar principle, the degree-corrected Laplacian was introduced by Chaudhuri *et al.*, that targets deficiencies associated with graphs that have strong degree heterogeneity [32]. They look primarily at Extended Planted Partition models, which extend the Stochastic Block Model (SBM) by adding a per-node parameter that contributes to the edge-presence probability. While classic SBMs have strong degree homogeneity, this additional parameter allows node degree to vary considerably more, even within the same cluster. Based on the finding by Mihail and Papadimitriou that graphs with Zipf-like degree distribution inordinately skews the resulting eigenspace [33], Chaudhuri *et al.* propose a form of regularisation for the degree matrix. This is achieved by adding a scaled copy of the identity matrix to D before proceeding with spectral clustering methods on the resultant L^{RW}. Closely related work by Qin and Rohe defines the regularised Laplacian as

$$L^{\tau} = (D + \tau I)^{-\frac{1}{2}} L (D + \tau I)^{-\frac{1}{2}} ,$$

where τ is some non-negative constant [34]. They go further by deriving some sensible bounds for the value of τ, but observe that good results are achieved by setting $\tau = \mathbb{E}(d)$.

In our experiments of Section6.1.1, we analyse the effects different Laplacian variants, namely L, L^{sym} and L^{RW} and their τ-regularised versions $L^{\tau-sym}$ and $L^{\tau-RW}$, setting τ as suggested by Qin and Rohe. We also include the results of running our algorithm using the unmodified W for comparison.

Preprocessing runtime

The time spent performing these preprocessing steps is not inconsiderable, but is primarily determined by the time taken to calculate the eigenvalues and eigenvectors of the laplacian for the original input G. Pan and Chen show that for general matrices of size $n \times n$, this operation scales in the worst case as $\mathcal{O}(n^3 + (n\log^2 n)\log b)$, where 2^{-b} represents the tolerance [35]. However, we can take advantage of two key characteristics for our spectral decomposition. Firstly, we are concerned with Laplacian matrices of graphs, which are typically sparse, and able to be expressed using formats such as Compressed Sparse Row (CSR) [36]. Though exact costs vary depending on the structure of the graph, this allows for matrix-vector multiplication to be performed in close to $\mathcal{O}(n)$ time, rather than $\mathcal{O}(n^2)$. Secondly, we are only seeking a handful of eigenvectors, ordered by the magnitude of their eigenvalues, which can be found efficiently using the ARPACK software package developed by Lehoucq *et al.* [37]. This avoids calculation of any inverses, which as noted by Hackbusch is $\mathcal{O}(n^3)$-complex, and typically produces a dense output, even when the input is sparse, leading to $\mathcal{O}(n^2)$-complexity for subsequent matrix-vector multiplication [38]. Instead, ARPACK employs the Arnoldi process for iteratively approximating eigenvalues and eigenvectors to within a desired tolerance, for which the execution time is determined primarily by the cost of matrix-vector multiplication. In the case of of symmetric matrices – for example, Laplacian matrices of undirected graphs – this can be further improved by using the Lanczos process instead [39, Ch. 4.5].

4.2 Feature extraction

Following the spectral decomposition of Section 4.1.1, we can produce a matrix of structure-derived node features. We parametrise which feature are included according to: the number of times the input G is recursively partitioned; the number of eigenvectors $\mathbf{z}_1, \ldots, \mathbf{z}_n$ to include; which partitioning method to use, (*median*, *sign* or *ncut*); whether to use zero, one or multiple virtual nodes, and whether to weight these. Additionally, we can include the binary partitioning vectors generated at each split, indicating precisely to which partition each node was eventually assigned.

We may also choose to utilise the eigenvalues themselves. These will be identical for all nodes assigned to the same partition, but do give some insight into the structure of the resulting partition. As we established in Chapter 3, the multiplicity of λ_0, which is always 0, indicates the number of connected components. However, the value of the smallest nonzero eigenvalue, known as the Cheeger constant $h(G)$, can indicate the *conductance* of G, i.e. how 'bottlenecked' it is [40, 41]. Small values of $h(G)$ occur when there is a substantial bottleneck, e.g. an edge between two otherwise disconnected components, with the logical extension that when this edge is removed, the components become separated and the eigenvalue is driven to 0.

4.3 Local node features

In order to verify that the eigenfeatures we have described are helpful for solving a given task, we form an alternative feature matrix based on localised node features. Those provided include, for a given node u:

- Degree centrality: the fraction of all nodes connected to u, i.e. d_u/N
- Clustering coefficient: the fraction of all possible triangles through u which are closed, i.e. the fraction of all neighbours of u which share an edge.
- Square clustering coefficient: the fraction of all possible squares through u, i.e. the fraction of all neighbours of u which share an edge to a fourth node that is not u.
- Average neighbour degree: $\frac{d(i)}{|\mathcal{N}(u)|}\ \forall\, i \in \mathcal{N}(u)$, where the neighbourhood $\mathcal{N}(i)$ of node i is the set of all nodes with an edge connected to i;
- Closeness centrality: the reciprocal of the average length of the shortest path to each node in the connected graph. Can be calculated as $(n-1)/\sum_{v \in V} dist(u,v)$, where $dist(u,v)$ refers to the shortest path between u and v.
- Eccentricity: the maximum shortest path distance to any other node, i.e. $max\{dist(u,v)\ \forall\, v \in V\}$.
- A binary variable indicating whether u is part of the periphery of G, i.e. no other node has an eccentricity greater than that of u.

- A binary variable indicating if u is at the centre of G, with no other node having an eccentricity less than u

Note that the latter four features all rely on having pre-computed the matrix of shortest paths between all pairs of nodes in G. As such, these incorporate global information about the graph, so for the majority of our experiments we limit the local features to the first four.

4.4 Model design

Several varieties of model are compared to see how well they perform clustering in a supervised setting. We first test the MLP architecture, transforming an input feature matrix $\mathbf{X} \in \mathbb{R}^{|N| \times f_0}$ through multiple layers to produce an output $\hat{\mathbf{Y}} \in \mathbb{R}^{|N| \times k}$. Note that predictions for each node are independent of one another, and so there is no opportunity for the network to determine with certainty which clusters other nodes have been assigned to, either locally or globally.

We next examine a GNN, that uses message passing layers to aggregate information in each node's local neighbourhood. The specific architecture used employs graph convolution layers of the form

$$\mathbf{x}_i^{(t)} = \sigma\left(\mathbf{x}_i^{(t-1)} \cdot W_1^{(t-1)} + \sum_{j \in \mathcal{N}(i)} a_{i,j}\, \mathbf{x}_j^{(t-1)} \cdot W_2^{(t-1)}\right) \quad ,$$

where σ is a function that introduces nonlinearity, typically ReLU; $\mathbf{x}_i^{(t)}$ is the input vector for node i at layer t; $W_1^{(t-1)}$, $W_2^{(t-1)} \in \mathbb{R}^{f_{(t-1)} \times f_t}$ are matrices of learnable weights; and $a_{i,j}$ is the weight of such edges [29]. Again, this model has to rely on spectral features for global information, and cannot generally know about cluster assignments of distant nodes. However, it can make more informed judgements about a node's local region, based on the information of its neighbours.

A shortcoming of both these approaches is the need to specify the size of the output vector in advance, imposing a constraint on the maximum number of clusters that can ever be used. In certain circumstances where we have prior information about the structure of a graph, this may not be a problem – we know, for instance, that the proteins found in human bodies are composed of at most 22 different amino acids [42]. In general however, this constraint restricts the model's ability to generalise to unseen graphs drawn from distributions with a greater number of clusters.

To target such situations, we re-frame the task of directly assigning nodes to clusters as an edge-prediction problem. We design a model that indicates whether two nodes belong in the same cluster by predicting the existence of an edge between them. This results in a new graph where each cluster is represented by a single disconnected component. If we invert this and instead predict whether to *remove* an edge from the original input G, this guarantees that any clusters will be contiguous.

For edge-based clustering, we again use a GNN to generate node representations. We follow this by concatenating the two node representations for each edge (u, v) to produce edge representations

$$\mathbf{e}_{u,v} = (\mathbf{x}_u^T \oplus \mathbf{x}_v^T) \;\; \forall\, (u, v) \in E \quad ,$$

where $\mathbf{x}_i^T$ is the representation of node i in the final GNN layer and $\oplus$ denotes concatenation along the feature dimension.

We follow this by MLP layers of the form

$$\mathbf{e}_{u,v}^{(s)} = \sigma\left(\mathbf{e}_{u,v}^{(s-1)} \cdot W_3^{(s-1)}\right) \quad ,$$

where $\mathbf{e}_{u,v}^{(s)}$ is the representation of the edge (u, v) at layer s; σ applies a suitable element-wise nonlinearity; and $W_3^{(s-1)} \in \mathbb{R}^{f_{(s-1)} \times f_s}$ are again learnable weight matrices. For the output layer S, $W_3^{(S-1)} \in \mathbb{R}^{f_{(S-1)} \times 2}$, and the *softmax* function is used for σ.

4.5 Efficient output interpretation

In certain domains we can perform supervised clustering, if we already have example of desirable clusters, such as secondary structures of proteins, or well-defined objects in images. We can use these to verify how well the model performs clustering, but must be mindful that any assignment of nodes among k clusters have $k!$ equivalent assignments. The simplest approach, outlined by Chen, Li and Bruna, is to maximise accuracy over every possible permutation [16]. However, as they observe, evaluating this for large k quickly becomes impractical; they only carried out experiments for $k \leqslant 5$.

Fortunately a more efficient solution exists, if we approach the issue as a bipartite matching problem. Suppose we have an input graph G with nodes N, and a vector of labels $\mathbf{y}$ indicating the desired way to assign each node to one of k clusters, such that $\mathbf{y}_i \in \{1, \ldots, k\}$. We design a model to output some clustering predictions as a similarly shaped vector $\hat{\mathbf{y}}$. Our aim is to find the optimal permutation of the *values* of $\hat{\mathbf{y}}$ that maximises the similarity between $\hat{\mathbf{y}}$ and $\mathbf{y}$; the number of correct predictions is then simply the frequency with which $\hat{\mathbf{y}}_i = \mathbf{y}_i, \ \forall i \in N$.

First, we create an evenly-sized bipartite graph B with $2k$ nodes, denoted $Y_1, \ldots, Y_k, \hat{Y}_1, \ldots, \hat{Y}_k$. Each of these nodes in B is assigned a list of nodes derived from either the labels or the model output, such that Y_c is assigned a list of all nodes labelled as belonging to cluster c, denoted $nodes_{Yc}$, and $\hat{Y}_c$ is assigned a list of all nodes which the model predicts as belonging to cluster c, denoted $nodes_{\hat{Y}c}$. We then proceed to add weighted edges between every pair of nodes $(Y_i, \ \hat{Y}_j)$ for $i, j \in 1, \ldots, k$. The weight of each edge is equal to the size of the intersection of the relevant lists, i.e. $|\,nodes_{Yi} \cap nodes_{\hat{Y}j}\,|$. Clearly, if $|\,nodes_{Yi} \cap nodes_{\hat{Y}j}\,| = |\,nodes_{Yi}\,| = |\,nodes_{\hat{Y}j}\,|$, this indicates an exact match between the partitions, any permutation of the values of $\hat{\mathbf{y}}$ should replace cluster i with cluster j. We wish to recover the optimal one-to-one mapping between the values of $\hat{\mathbf{y}}$ and $\mathbf{y}$, that maximises the intersection between predictions and labels; this is now equivalent to solving the maximal matching of B. Using an algorithm such as the Hungarian method, this can be solved in the worst case in $\mathcal{O}(k^3)$ time [43].

4.6 Loss functions

We primarily train models using the negative log-likelihood (NLL) loss function [44, pp. 131-133]. However, we also experiment with two alternate loss functions, as well as two regularisation functions, all of which we now describe.

4.6.1 Silhouette-like loss

Even with the efficiencies outlined in Section 4.5, as the number of possible clusters increases this still begins to dominate the training and evaluation time for models. We design a more suitable loss function that makes use of the Silhouette Coefficient s, calculated for every node n as

$$s = \frac{b - a}{max(a, b)} \quad ,$$

where a is the mean distance between n and each other node in its cluster, and b is the mean distance between n and each node in next nearest cluster [45]. The Silhoette Coefficient for the graph is then the mean of all the individual node coefficients.

The most natural distance metric is the length of the shortest path between two nodes, which is calculated in $\mathcal{O}(N^3)$ time using the Floyd-Warshall algorithm [46] as a preprocessing step and stored with the sample as a shortest-path distance matrix $D_{SP} \in \mathbb{R}^{n \times n}$. We can then take the matrix product $D_{NC} = D_{SP} \cdot C \in \mathbb{R}^{n \times c}$ to produce a matrix representing the distance between each node-cluster pair.

The elementwise product $D_{NC\text{-}weighted} = C \odot D_{NC}$ represents the mean distance of a source node to every other node in a target cluster, weighted by the affinity for the source node to be assigned to the target cluster. Summing over all elements of $D_{NC\text{-}weighted}$ gives a representation of intra-cluster distances a that accommodates for nodes being strongly or weakly assigned to certain clusters.

To calculate the between-cluster distances b in the same manner, we instead calculate $D_{NC\text{-}inverse} = (1 - C) \odot D_{NC}$, which provides the mean distance from a source to a target cluster, weighted by the likelihood of it *not* being assigned to that cluster. Thus the distance to its 'nearest' cluster – the one to which it should have the strongest affinity – contributes comparatively less, than the distance to other clusters. The value for b is then calculated as the mean over every value of $D_{NC\text{-}inverse}$. An alternative was tried out by taking only the smallest N distances in $D_{NC\text{-}inverse}$ to take a mean over, (excluding the very smallest), in order to adhere more strictly to the concept of a 'next-nearest cluster'. However, better results were obtained by averaging the distances to all other clusters to obtain b.

Note also that the range for s is translated from $[-1, 1]$ to $[0, 2]$ when deployed, in order to avoid the relative importance of the loss changing as the magnitude decreases then increases again.

4.6.2 Modularity loss

Given a highly modular graph, if we sample random edge from a source to target node, we would expect that both source and target nodes lie in the same cluster more frequently than straddle two clusters. To incorporate this as a regularisation term, we start with Newman's modularity, calculated as

$$Q = \sum_{i=1}^{C} (e_{i,i} - a_{i,\cdot}^2) \quad ,$$

where $e_{i,i}$ represents the proportion of edges contained entirely inside cluster i, and $a_{i,\cdot}^2$ is the proportion of edges partially *or* entirely within cluster i, with both values normalised by dividing by the total number of edges in the graph [2]. The modularity Q of the assignment for the whole graph is the sum of each cluster-specific modularity.

Next we need to reframe this metric to accommodate the varying affinities for node-cluster assignments. We achieve this by calculating C^TAC, which results in a matrix $M \in \mathbb{R}^{c\times c}$ that sums all the edges in the adjacency matrix A, weighted such that $M_{i,j} = \sum_y \sum_x A_{x,y} \cdot C_{i,x} \cdot C_{j,y}$, i.e. each entry $M_{i,j}$ is a sum of all edges between nodes in clusters i and j, weighted by those nodes' affinities for said clusters. This is equivalent to $a_{i,\cdot}$ defined above, for each cluster, with the values for $e_{i,i}$ lying along the diagonal of M.

For similar reasons to the Silhouette-like loss, we take the mean modularity over clusters rather than the sum, and normalise by dividing by $max(a, e)$ instead of by the total number of edges in the graph, (again translating the domain from $[-1, 1]$ to $[0, 1]$).

4.6.3 Regularisation

Cluster assignment entropy

Models were found to have a tendency, especially early on, to only weakly assign nodes to clusters. To overcome this, the entropy of the output layer for each node is averaged, and added to the loss. Left by itself with no other loss function, this 'cluster assignment entropy' will quickly default to assigning every node to a single cluster, and is therefore used sparingly, but in small doses can speed up the learning process significantly. This regularisation is similarly employed by DiffPool [10].

Total cluster entropy

While the cluster assignment entropy is calculated by taking entropy over each row of C, the total cluster entropy can be found by first summing each column of C and then calculating the entropy over the resulting vector. This vector can be considered a measure of the relative size of each cluster, and hence has high entropy when clusters are all being used in roughly equal proportions. Adding this to the loss has the effect of driving down the number of different clusters used, on average, and helps avoid cluster assignments where small, dense graphs are ascribed multiple clusters, when a single cluster might be more sensible.

Chapter 5

Datasets

We outline here the theory and design of five synthetic datasets used for the initial evaluation in Section 6.1. We then briefly describe the three real-world datasets that will be used for benchmarking graph classification and community detection models in Section 6.2.

5.1 Synthetic data

The artificial data is constructed from four families of graph, which all contain natural clusters to varying degrees. We constrain our investigation to undirected graphs composed of a single connected component, and describe their construction presently.

Stochastic Block Models

We generate SBMs by first choosing a number of nodes N and number of clusters C. Nodes are first assigned to clusters using a probability vector $\mathbf{c}$, with elements c_i indicating the probability of a node being assigned to cluster i for $i \in 1, \ldots, C$. Edge presence between every pair of nodes $(u \in c_i,\ v \in c_j)$ is then determined according to a symmetric matrix $W \in \mathbb{R}^{C \times C}$, where the entry $W_{i,j}$ indicates the likelihood of an edge between u and v. It follows that the diagonal entries of W describe the within-cluster edge density.

There are circumstances when it may not be possible to identify the community structure of SBM. Specifically, we can consider the expected adjacency matrix for each cluster $\mathbb{E}(W)$, formed by taking the elementwise product $W \odot \mathbf{c}\mathbf{c}^T$. The expected degree of each cluster is found by multiplying the elements of $\mathbb{E}(W)$ by N. We denote the eigenvalues μ_i of $\mathbb{E}(W)$ in *decreasing* order of magnitude $|\mu_1| \geq |\mu_2| \geq \cdots \geq |\mu_C|$, where μ_1 also indicates the expected node degree of the graph. Decelle *et al.* then conjecture that the true community distribution can be theoretically identified if and only if $\mu_2^2 > \mu_1$ [47, 48]. Decelle *et al.* employ belief propagation to recover the distribution, though spectral approaches such as that proposed by McSherry are also effective [49, 50].

In the case of symmetric SBMs, described by Abbe [51] as the case where $W_{i,j} = p_A \ \forall \ i = j$ and $W_{i,j} = p_B \ \forall \ i \neq j$, we can determine the values of μ_1 and μ_2 as

$$\mu_1 = \frac{N}{C}\left[p_A + (c-1)p_B\right] \ ,$$

$$\mu_2 = \frac{N}{C}(p_A - p_B) \ .$$

Given p_A, p_B and C, this leads to the following expression that indicates the size of the graph that would be required under Decelle's *et al.* conjecture to determine the community structure as

$$N > \frac{C(p_A + (c-1)p_B)}{(p_A - p_B)^2} \ .$$

For the case where $C = 2$ and $\mathbf{c} = [\frac{1}{2}, \frac{1}{2}]$, Mossel, Neeman and Sly give separate proofs of both components of Decelle's *et al.* conjecture [52, 53]; an independent proof of the positive component was also demonstrated by Massoulié [54].

In order to draw a sufficiently varied dataset of SBMs for the experiments in Section 6.1, we specify several parameters. We limit the number of nodes and clusters to between 20–100 and 2–5 respectively. The prior probability over clusters is specified such that $c_i \geq \frac{2}{3} c_j, \forall i, j \in 1, \dots, C$. We constrain the edge presence within-cluster probabilities to $[0.2, 0.35]$ and the between-cluster probabilities to $[0.03, 0.05]$. No self-loops are added and resultant samples are rejected if the degree of any node is zero.

These parameters can in some instances produce samples for which the complete recovery of the true distribution is theoretically impossible. In the worst case $C = 5$, $p_A = 0.2$, $p_B = 0.05$ leads to a requirement that $N > 88.9$ to recover the distribution, which would typically not be met given that $N \leq 100$. In general however there is a useful amount of information available from just the node degree that can be used to make clustering decisions.

Hierarchical Networks

Analysis by Ravasz and Barabási revealed that many real-world networks possess two important properties: a scale-free topology, and a highly modular structure [55]. The former tells us that the degree distribution of nodes follows a power law, with the probability of a random node having degree k being distributed as $P(k) \sim k^{-\gamma}$. The latter can be inferred using the clustering coefficient of a node i

$$C_i = \frac{2n_i}{k_i(k_i - 1)} \quad ,$$

where n_i is the number of edges between the k_i neighbours of i. Ravasz and Barabási find these properties in a variety of datasets, such as the IMDB Actor Network, a language network with edges between words listed as synonyms of one another, and a map of the www.nd.edu domain with hyperlinks indicating edges between pages. They conclude the common feature is a hierarchical organisation, where the clustering coefficient of a node given its degree $C(k)$ scales as k^{-1}. Intriguingly, they analyse two *geographically*-based networks, namely the internet at the router level and the power grid of the Western United States, and find that $C(k)$ is independent of k, suggesting these networks do *not* possess a hierarchical structure. This makes sense when we consider that there is a higher physical cost associated with introducing edges between geographically distant nodes, that is not present when dealing with adding edges in an abstract domain such as language.

In order to better model hierarchically organised networks, Ravasz and Barabási suggest constructing networks in an iterative manner. We follow the same approach, generating samples by beginning with a small, randomly generated motif, with a single 'hub' node. This motif is then copied a number of times according to a branching factor b, with all non-hub nodes connected to their parent hub. This entire process can then be repeated; we refer to the number of iterations as the depth d. A canonical example with a starting motif size of 5, $b = 4$ and $d = 2$ is shown in Figure 5.1. Note 5-node communities are fully-connected; edges between diagonally opposite nodes omitted for clarity. In our experiments, we generate two datasets, with labels provided for the first (HN_1) and second (HN_2) layers; in the case of Figure 5.1, each of the five easily-identifiable 25-node clusters is labelled with a different community identifier at the first layer, and each of the 5-node clusters is labelled at the second layer. For HN_1, we vary the size of the initial motifs from 4–7 nodes, the branching factor from 2–8, and perform either 1 or 2 passes of the pattern copying. For HN_2, the motifs are 4–6 nodes in size, the branching factor is between 2–3, and all samples are generated using two passes.

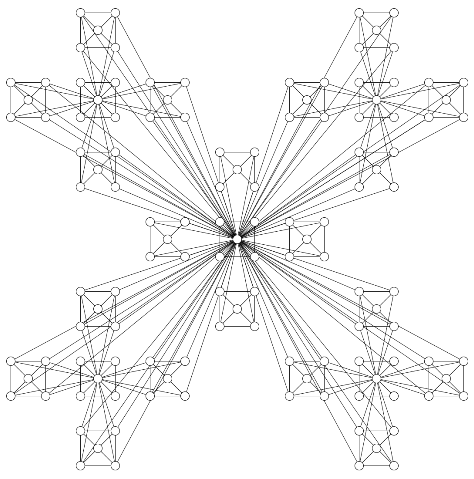
Figure 5.1: Canonical hierarchical network, reproduced from [55].

Expanded Node

Motivated by the desire to study smaller, less well-defined communities, we generate EN graphs in the following manner:

1. Generate a random undirected graph G_{init} with N nodes, with probability of each edge presence set to ensure an expected node degree $\mathbb{E}(d_1)$.

2. Visit each node $i \in 1, \ldots, N$ in turn, randomly generate another subgraph G_i of size N_i and edge presence probability set to ensure expected node degree $\mathbb{E}(d_2)$.

3. Remove node i from G_{init} and any edges $(i, j) \in G_{init}$.

4. For each previously existing edge (i, j), choose a random node $u \in G_i$ and add the edge (u, j) to G_{init}.

The community labels are set such that all nodes in subgraph G_i share the community identifier i. Although it is possible to choose only to expand nodes with a certain probability, this would result in communities of size 1, which we do not explore. Most of our experiments are carried out on EN graphs with the size for G_{init} in the range $[5, 20]$ and $\mathbb{E}(d_1) = 2.5$. The subgraph sizes are in $[6, 12]$ and $\mathbb{E}(d_2) = 3.0$.

Shared Motifs

We also develop a modified version of EN networks, which we refer to as 'Shared Motifs' networks. These are designed to mimic the repeated appearance of motifs, another common real-world pattern, such as the proteins referred to in Section 4.4. To achieve this, we first generate a pool of motifs that will be shared by every sample in the dataset. We generate samples by following the routine for generating the EN networks, but expand nodes using a randomly selected motif from the pool, instead of creating a completely new subgraph. The motif pool can be initialised using random graphs, as generated for the ENs, but may also contain common patterns, such as cliques or stars.

For the experiments here, we set the size of G_{init} from the range $[5, 20]$, with $\mathbb{E}(d_1) = 2.0$. For the pool, the motif sizes are chosen from the range $[5, 10]$, with $\mathbb{E}(d_2) = 3.5$. We set the pool size to 20 and do not use cliques or stars,

Table 5.1: Structural statistics for the 4 artificial network types. Reported values are means over all samples in test set.

	SBM	HN_1	HN_2	EN	SM
# nodes	60	137	109	106	93
# edges	252	345	254	184	149
density	.127	.084	.054	.039	.042
clustering coeff.	.192	.549	.489	.293	.244
clustering coeff.$_{sq}$	.047	.327	.271	.206	.208
diameter	4.88	6.17	8.33	14.8	17.4
deg. assort.	.055	-.236	-.230	-.052	.296

but do include rings, (for which $\mathbb{E}(d_2)$ is not required).

5.1.1 Comparison of synthetic data sets

Summary statistics for each of the synthetic datasets are presented in Table 5.1, which shows mean values over all samples. Both forms of the clustering coefficient are calculated as mean values over all nodes for each sample. We also report the degree assortativity, a measure of node degree similarity [56, 57].

We observe the significantly higher clustering coefficients for Hierarchical Networks. As shown by Ravasz and Barabási, the per-node clustering coefficient scales inversely with degree. In Figure 5.1, the node at the centre has very high degree but no closed triangles at all, and hence a clustering coefficient of 0. Conversely, the non-hub nodes in each of the 5-node communities have all of their triangles closed, leading to a clustering coefficient of 1 – note that diagonal links in these communities are present but not shown for clarity. As these nodes dominate the graph, this drives up the clustering coefficient. Similar observations can be made about square clustering.

5.2 Benchmark datasets

We use the Cora and CiteSeer citation networks to assess the real-world performance of the model. The Cora dataset comprises a single graph of 2,708 nodes and 5,429 edges [58]. Nodes represent academic papers, and an edge between two papers indicates that one references the other. Each node has 1,432 binary attributes indicating to the most frequent words in the paper, and is assigned one of 7 labels indicating the academic sub-field for the paper, e.g. ‘Neural Networks’ or ‘Probabilistic Methods’. The task is then to use the attributes and citation information to predict the community label. The CiteSeer dataset is structured similarly, with 3,312 nodes, 4,732 edges and 6 classes [59].

Additionally, the ENZYMES dataset created by Borgwardt *et al.* is used as a graph classification task [60]. This consists of 600 enzyme structures with an average of 32.6 and 62.1 nodes and edges respectively, with 18 node attributes. The task is to classify each input as one of 6 enzyme classes, and is a widely used benchmark for graph classification.

Chapter 6

Evaluation

6.1 Experiments on Synthetic Data

We compare four approaches for learning the correct assignment of nodes to clusters. Two models use the message-passing architecture outlined in Section 4.4, while two use a straightforward MLP network that classifies each node independently. One each of these pairs uses the preprocessed spectral information, with the other limited to using the four locally-derived node features not dependent on the distance matrix, described in Section 4.3.

Results are summarised in Table 6.1. All datasets used 600 training samples, (120 of which were used for validation), with a further test set of 120 unseen samples for evaluating performance. Models were trained for 60 epochs, with early termination in place if no improvement in validation performance occurred in the previous 25 epochs. Accuracies are based on the number of correctly classified nodes obtained over the full test set, during the epoch associated with the best validation performance. Each run was repeated 3 times and the mean accuracy reported.

We observe that when restricted to the localised features for the SBM and HN data, a GNN conveys no benefit over the MLP architecture. As the MLP classifies each node separately, we can infer that there is little infor-

Table 6.1: Baseline performance for each model-dataset pair.

	SBM	HN_1	HN_2	EN	SM
MLP-local	.559	.203	.140	.303	.328
MLP-spectral	.752	.748	.609	.635	.618
GNN-local	.520	.206	.109	.541	.476
GNN-spectral	.817	.820	.690	.691	.690

mation that can be obtained from a node's neighbours, for the purpose of informing clustering decisions. This is unsurprising when we consider that the construction of SBM and HN graphs necessarily results in most nodes being highly similar. Particularly for HNs, when node features do differ, this does not provide any usable information about the likely communities, and this causes the GNN-local model to struggle badly, especially so for HN_2. See Figures 6.1 and 6.2 for examples of SBM and HN classification by the GNN with spectral features, (taken from the testing set). Figure 6.3 shows the attempt by GNN-local to classify an HN – note the complete inability to classify based on the topographical location, instead resorting to classifying nodes roughly by their degree and distance from the centre.

Better predictions were achieved when the spectral information about global structure was provided, even when nodes are classified independently. However, the GNN was able to refine its knowledge about each node by aggregating information from the surrounding neighbourhood and make consequently better predictions.

For the EN and SM networks, the GNN fared much better with the local features. Examples of predictions are shown in Figures 6.4 and 6.5.

6.1.1 Comparison of Laplacian variations

We carry out experiments with spectral features derived from the Laplacian variations discussed in Section 4.1.1. We use the same train and test sets as the baseline experiments, and model structure and hyperparameters are left

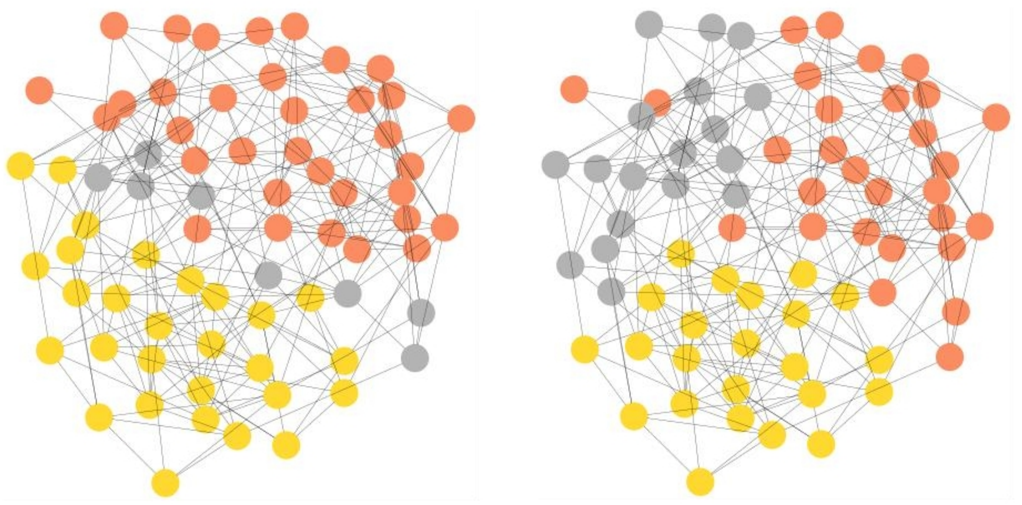

Figure 6.1: Left: GNN-spectral classification of SBM. Right: ground truth labels. The model identifies the correct number of clusters, and does a reasonable job of community prediction.

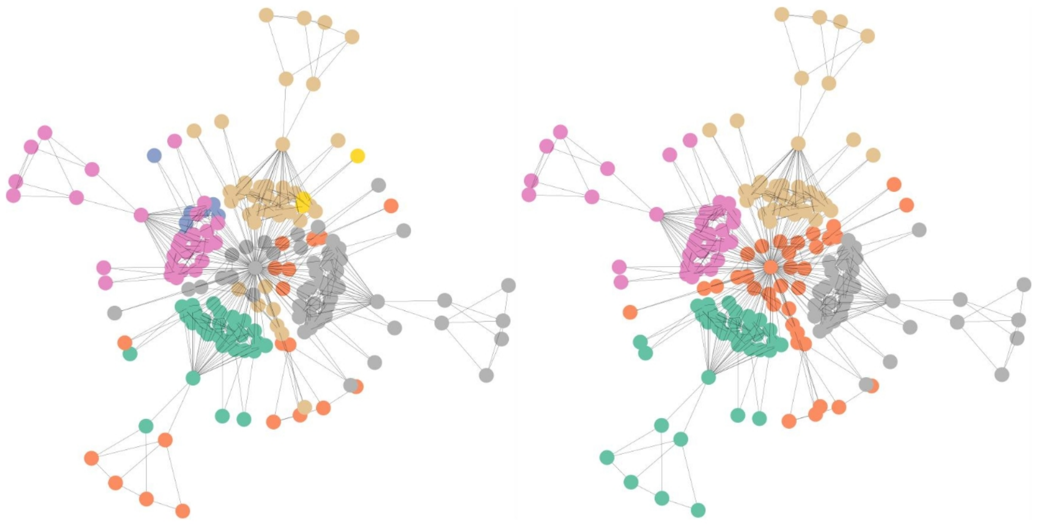

Figure 6.2: Left: GNN-spectral classification of HN with branching factor = 4, depth = 2 and an initial motif of 7 nodes. Right: ground truth labels for depth = 1. The model inadvertently introduces additional blue and yellow communities.

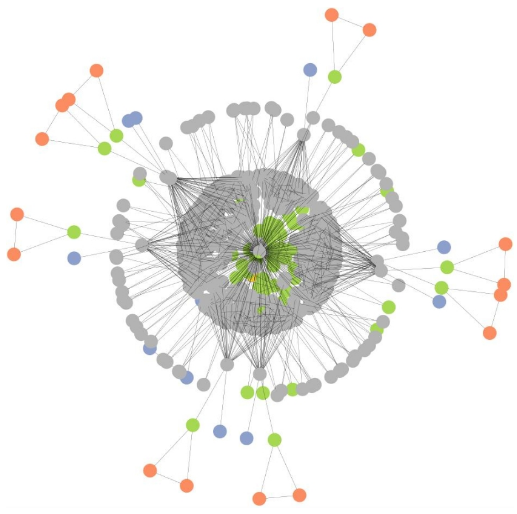

Figure 6.3: The GNN with local features makes poor clustering decisions based on node degree and distance from centre.

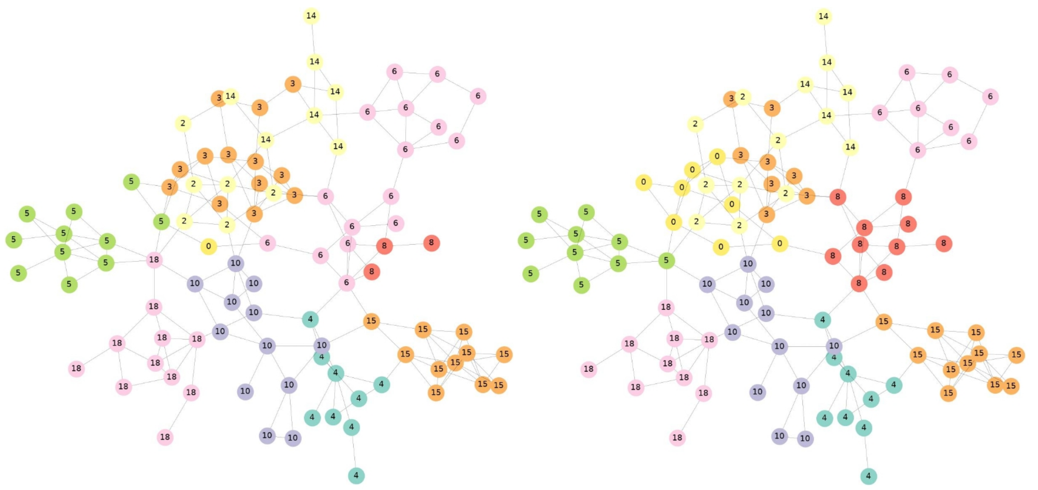

Figure 6.4: Predictions for an EN sample (left), with ground truth communities (right).

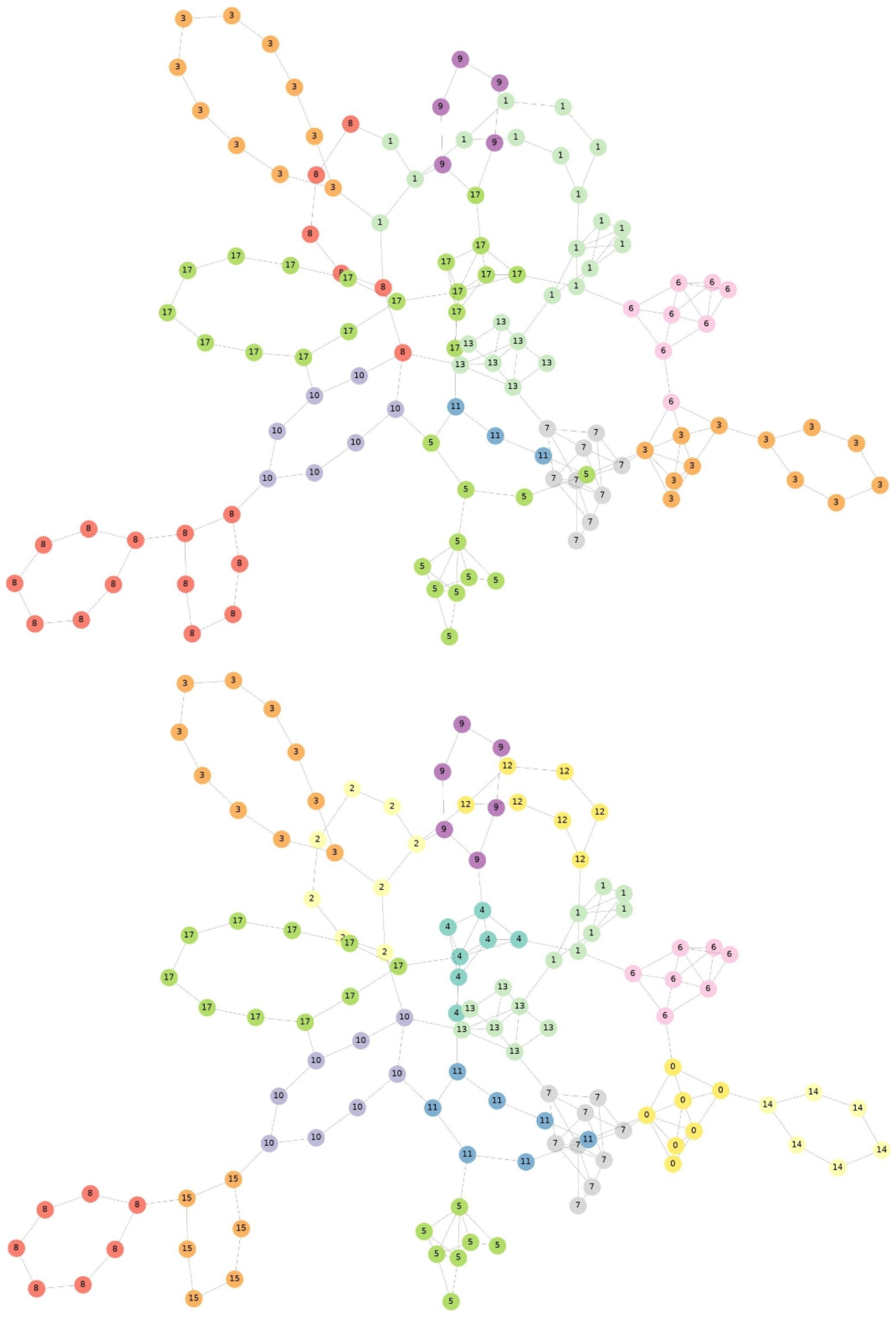

Figure 6.5: Predictions for SM sample (top), and labels (bottom). Note the presence of ring motifs results in a high frequency of nodes with degree 2.

Table 6.2: Comparison of different Laplacian variants

Variant	GNN					MLP				
	SBM	HN_1	HN_2	EN	SM	SBM	HN_1	HN_2	EN	SM
L	.817	.820	.690	.691	.690	.752	.748	.609	.635	.618
L^{sym}	.845	.812	.685	.716	.665	.810	.765	.606	.620	.624
L^{RW}	.843	.831	.693	.746	.713	.827	.752	.599	.725	.708
$L^{\tau\text{-}sym}$	.865	.806	.723	.729	.697	.829	.807	.671	.635	.612
$L^{\tau\text{-}RW}$	.871	.816	.723	.744	.751	.835	.815	.675	.654	.629

unchanged. Results are presented in Table 6.2.

We see that the L^{RW} variation generally achieved the better performance for both models, but the improvement over L was particularly marked for the MLP model, especially for EN and SM data. The key difference between L^{sym} and L^{RW} is that the former normalises elements by source and destination degree, producing a symmetric matrix with consequently orthogonal eigenvectors. L^{RW} normalises by the degree of the source node only, and the resultant non-orthogonal eigenvectors include an element of redundancy, which may be better utilised by models with reasonably high rates of dropout. In the context of EN performance, it is also noting the EN test set average node degree $\mathbb{E}(d) = 3.47$ is less than the $\mathbb{E}(d) = 5$ at which Amini *et al.* observed the difficulties of using L^{sym}, and this result appears to back up that claim.

For the HN_2 data, both τ-regularised Laplacians achieved a notable improvement. This is in line with the expectations that it should work well in samples with Zipf-like degree distribution and relatively heterogeneous node degree. See also the muted improvements over the non-regularised counterparts for the EN data, which in contrast has greater homogeneity of node degree.

Table 6.3: Comparison of virtual node configurations.

V-nodes	Weight	GNN					MLP				
		SBM	HN_1	HN_2	EN	SM	SBM	HN_1	HN_2	EN	SM
none	—	.817	.820	.690	.691	.690	.752	.748	.609	.635	.618
single	1	.812	.801	.666	.772	.768	.761	.700	.451	.585	.608
single	$\sum$	.814	.868	.673	.766	.751	.762	.779	.445	.591	.607
mult.	1	.794	.806	.640	.757	.762	.750	.697	.423	.579	.601
mult.	$\sum$	.793	.865	.637	.757	.764	.744	.772	.435	.597	.593

6.1.2 Virtual node experiments

We test each configuration of the virtual node methods. Either a single virtual node representing all non-partition nodes was added prior to computing eigenfeatures, or added one node for each other connected partition. We experiment with summing all the edges to the other partitions, as well as simply assigning each edge a weight of 1.

For the GNN, the presence of any virtual node led to a substantial improvement for the EN and SM data. The lack of any improvement for SBMs may simply be due to the fact that partitioning SBMs is less likely to result in disconnected components, and so adding a virtual nodes does not greatly change the structure of subgraphs. For HNs, accuracy improvements were only realised when the edge weights to the virtual node were summed. This may be explained by the fact when virtual nodes are added to a partition, the aim is to try to restore some of structural properties of the parent graph. As we have noted, HNs have a Zipf-like degree distribution, and therefore clipping the virtual node edge weights to 1 can result in a more significant distortion of the subgraph's connectivity.

The addition of any virtual node did not demonstrably help the MLP with the task, and the impact was particularly harmful with HN_2. This provides one of the clearest indications that the GNN learns a deeper association between the eigenfeatures and the computation graph, compared with the MLP. The addition of the virtual node ensures that all partitions are com-

Table 6.4: Comparison of split methods

Method	GNN					MLP				
	SBM	HN_1	HN_2	EN	SM	SBM	HN_1	HN_2	EN	SM
median	.817	.820	.690	.691	.690	.752	.748	.609	.635	.625
sign	.790	.710	.505	.763	.329	.753	.706	.457	.739	.135
ncut	.788	.644	.585	.770	.495	.789	.685	.516	.677	.200

posed of a single connected component, and as such the eigenfeatures are more continuous. While the MLP has no problem with disjoint eigenfeatures, due to classifying the nodes independently, this can cause problems for the GNN, which may have to reconcile how two neighbouring nodes can have unexpectedly disparate eigenfeatures. Conversely, the discontinuities in features may actually *help* the MLP, as this provides a more defined signal in the absence of information from neighbouring nodes, whereas the GNN is able to deduce information about the topographical location of a node from its neighbourhood. This is particularly apparent for the EN data, which significantly benefited from the addition of any virtual node.

6.1.3 Comparison of split methods

Two further experiments are carried out, using the *sign* and *ncut* procedures to assign nodes to each partition based on the Fiedler vector. Parameters and datasets are again unchanged from the baseline, and accuracies reported in Table 6.4.

Prior to running the experiments, we hypothesised that the *median* approach would net the strongest results. Our justification considered that if a split method produced unequally-sized partitions, then the nodes repeatedly assigned to the smaller partition would be more likely to end up in a partition with only one or two other nodes, from which little useful spectral information can be derived. Hence, by choosing the median value of the Fiedler vector as the split point, every subsequent split maximises the size of each split. Even if the resultant subgraphs were disconnected, we reasoned that the $\mathbf{z}_1$ vector would indicate to which component each node would belong. Subsequent vectors $\mathbf{z_2}$, $\mathbf{z_3}$ etc. could then be used to project a useful dimension on each component, as discussed in Chapter 3.

The results indicate this prediction may have been misplaced. Although the best overall performance came from the *ncut* method combined with the GNN, the most striking improvement for both models occurred using *sign*. This suggests that it is more important that to partition the input such that the subgraphs are more likely to be connected, than to ensure equal-sized subgraphs.

We also note that for the GNN, the *ncut* method did not significantly outperform the others for any dataset. Depending on the size of the graph and number of split points to check, the preprocessing requirements for *ncut* are typically far greater, and may not be worth it when *sign* achieves a similar outcome.

6.1.4 Eigenfeature evaluation

Number of splits

Several tests were run to help quantify the effectiveness of the eigenfeatures, focusing only on the EN networks. We begin by looking at how many times we partition the sample into subgraphs, with results shown in Table 6.5. We observe that increasing the number of splits does not appear to improve accuracy, and past 4 splits appears to harm its efforts. However, the EN graphs also had an average of 12.5 clusters; as 4 splits divide the graph into 16 partitions, this may explain why the GNN performance peaked at this point. Certainly, with 8 splits, the resulting partitions are likely to be very small or single-node subgraphs, which convey no useful clustering information and contribute only noise, explaining the hit to performance.

Table 6.5: Varying number of splits for EN graphs.

# splits	GNN	MLP
2	.700	.655
4	.702	.650
6	.691	.635
8	.676	.636

Table 6.6: Varying number of eigenvectors for EN graphs.

# vectors	GNN	MLP
1	.593	.523
2	.638	.602
3	.652	.602
4	.654	.597
5	.660	.604
6	.673	.604

Number of eigenvectors

Next we try and clarify the effect of varying the number of eigenvectors included with each split. We drop the eigenvalues and partition assignment feature entirely, and vary the number is vectors from 1–6. Table 6.6 shows a clear trend for the GNN to leverage the additional information towards better performance. Past 2 vectors, however, there is no discernible benefit to the MLP. This suggests that the high-frequency information conveyed by higher-order eigenvectors is most useful in the context of a local neighbourhood, with the MLP only able make broader approximations from the low-frequency features.

Table 6.7: Varying number of eigenvalues for EN graphs.

# values	GNN	MLP
0	.685	.636
1	.691	.635
2	.691	.652

Number of eigenvalues

We included eigenvalues to see if it could help the model derive information about the likely structure of the subgraph a node was assigned to at a given depth, without having to deduce this indirectly from the partition assignment variable. As mentioned in Section 3, the λ_1 value can help describe how well connected a graph is. Hence, the model may be able to use this information to place more confidence in the eigenvector associated with with smaller λ_1.

Table 6.7 shows the results of experiments where we kept the original parameters from the baseline experiments, varying only the number of eigenvalues. We observe very marginal improvements in accuracy as the number of values is increased, but the improvement is very small for both models.

For completeness, we also test to see if clustering can be performed using only the eigenvalues. All nodes that are assigned to a given partition during a split share the same eigenvalue feature for that split, so in principle it should be possible to deduce the partitions a node has been assigned to at any depth from only the eigenvalues – assuming that the eigenvalues are unique. This could be considered a somewhat circuitous route to recovering the binary assignment variable, which we know contributes significantly to clustering decisions. However as Table 6.8 shows, in practice the effects of using only the eigenvalues leads to very poor performance.

Binary assignment variable

At every split depth when computing the eigenfeatures, we partition the graph into two. We include as a feature a binary variable indicating to which

Table 6.8: Using only eigenvalue inputs for EN graphs.

# values	GNN	MLP
1	.474	.376
2	.462	.388
3	.441	.376

Table 6.9: Using only the assignment variable for EN graphs.

# configuration	GNN	MLP
baseline	.691	.635
assignment only	.688	.506

partition each node was assigned, which can be used to uniquely identify every resulting subgraph at the lowest depth. We can compare results from Table 6.6, which did not use the binary assignment variable, with the results from Tables 6.5 and 6.7, which did. We observe that for both models, the only occasion where a configuration with the assignment variable was outperformed by one that did was when 0 eigenvectors were included. This suggests the assignment variable may be the most significant factor in determining run performance. To test this further, we run the model with *only* the assignment variable, and find that this configuration nearly reaches the baseline performance for the GNN (Table 6.9). The MLP takes a much more substantial performance hit when only supplied with the assignment variable, but this is unsurprising when we consider the relative difficulty of the task in this context, equivalent to independently assigning nodes to clusters based only on the similarity of binary strings.

Table 6.10: Using a single pass

# vecs	GNN					MLP				
	SBM	HN_1	HN_2	EN	SM	SBM	HN_1	HN_2	EN	SM
baseline	.817	.820	.690	.691	.690	.752	.748	.609	.635	.618
3	.802	.745	.351	.614	.589	.777	.804	.338	.607	.578
6	.791	.750	.412	.673	.636	.756	.793	.398	.648	.649
10	.789	.750	.494	.715	.663	.742	.793	.469	.655	.642
20	.803	.746	.650	.697	.635	.742	.731	.587	.662	.617

Single pass

We carry out experiments where we do not partition the input into subgraphs, and instead gather only eigenvectors of the original graph. Table 6.10 shows the results of this for 3, 6, 10 and 20 eigenvectors, for all five synthetic datasets. Neither eigenvalues or the binary assignment variable are included as features.

The GNN tended to perform worse when then eigenvectors of subgraphs are not used. We also note that this affected the HN dataset more significantly than the others, which emphasises the importance of the partitioning algorithm when working with hierarchical data. Results for the MLP architecture were less conclusive, but in general the lack of subgraph features was not felt as keenly, even for the HN data. We infer from this that the MLP approach struggles to learn useful concepts about the partitions in the way the GNN can.

We also note that in general, extending the number of eigenvectors past the 3–6 range did not usually correlate with improved performance. We interpret from this that higher frequency eigenvectors, for graphs of the size, appear to contribute mostly noise. The clear exception is the HN_2 dataset, for which many vectors were required to start returning to the baseline performance.

This appears to be related to the fact that communities for HN_2 are generally smaller and more numerous, and therefore the first few eigenvectors are too low-frequency to be of use. In contrast, these smaller communities appear to 'resonate' with the higher-frequency eigenvectors.

Loss regularisation

We carry out experiments using the Silhouette-like and modularity losses outlined in Section 4.6, again focusing on EN graphs. The λ parameters are set to 1 or 0 and every configuration tested, (we ignore the MLP-local model variation, which invariable performs badly). Results shown in Table 6.11. We find that using both these regularisation functions provides a small but consistent improvement in classification accuracy for all model variations.

Crucially, we also observe that in almost every case, addition of one or both of the regularisation results in a reduction in the NLL loss. Even where this does not yield a tangible improvement in accuracy, this does suggest that the additional terms help the model learn useful clustering information and perhaps generalise better.

Recall that neither of these functions require visibility of the ground truth labels, and can therefore be considered an unsupervised approach. As such, it is disappointing that better performance could not be achieved by dropping the NLL function entirely. Nevertheless, the GNN-spectral with NLL turned off comes close to matching the performance of the baseline GNN restricted to local features, demonstrating that they do bring a clear benefit by providing an alternative training signal for clustering.

We also perform experiments that vary the λ parameters for the TCE and CAE regularisation terms described in Section 4.6.3. Unfortunately, we found no benefit in using these; as Table 6.12 shows, they generally harmed the final accuracy, often significantly. This could be due to imposing an incorrect assumption about the likely community structure of the data. There may be benefit in using these to increase confidence in clustering decisions where this may be required for downstream processing tasks as in architectures

Table 6.11: Unsupervised loss functions on EN graphs.

λ_{NLL}	λ_{silh}	λ_{mod}	GNN-spectral		GNN-local		MLP-spectral	
			acc	nll	acc	nll	acc	nll
1	0	0	.691	1.537	.544	1.939	.635	3.101
0	1	0	.533	2.532	.454	2.781	.430	5.645
0	0	1	.498	2.562	.404	2.861	.198	6.540
1	0	1	.686	1.403	.544	1.935	.638	3.165
1	1	0	.696	1.486	.552	1.901	.669	2.852
0	1	1	.530	2.537	.501	2.477	.492	5.186
1	1	1	.710	1.277	.563	1.861	.674	2.896

such as DiffPool, but for predicting community membership itself, we see little reason to justify artificially enforcing a high degree of confidence, or otherwise manipulating the preferred number of clusters detected.

6.1.5 Varying the EN distribution

An analysis was carried out on how the overall structure of EN samples change as the distribution parameters are varied. We generate datasets with the values for both $\mathbb{E}(d_1)$ and $\mathbb{E}(d_2)$ in the range $[1, \dots, 7]$, with other parameters as before. Following this, we then test the performance of baseline models, (except for MLP-local), for $\mathbb{E}(d_1)$ and $\mathbb{E}(d_2)$ in the set $\{1, 4, 7\}$. We show summary statistics and the evaluation performance in Tables 6.13 and 6.14 respectively.

Predictably, all models improve as motif degree increases, and transitivity with it. However, as $\mathbb{E}(d_{init})$ was raised, reducing the diameter of the graphs, we expected spectral features to be less useful, due to the increased difficulty of solving the *ncut* problem. For lower values of $\mathbb{E}(d_{motif})$ this holds true, but we were surprised that for both spectral models with a sufficiently high $\mathbb{E}(d_{motif})$ this did not appear to impact performance. This suggests spectral

Table 6.12: Prediction regularisation on EN graphs.

TCE	CAE	GNN-spectral	GNN-local	MLP-spectral
0.0	0.0	.691	.541	.635
0.0	0.1	.683	.527	.656
0.0	0.5	.636	.509	.582
0.0	1.0	.605	.459	.537
0.1	0.0	.686	.531	.637
1.0	0.0	.675	.524	.630
10.0	0.0	.572	.460	.540

features are still highly useful for imposing spatial 'dimensions' on graphs with a smaller diameter, when nodes are reasonably well clustered.

6.1.6 Local features

Lastly, we assess the importance of the various local features used, again focusing on the EN networks. Table 6.15 shows the resulting performance when each of the local features is dropped. Also presented is the separate and combined performance when all 8 features are provided, including those dependent on the distance matrix. Using all does not convey any performance benefit. The most significant features are the measures of clustering coefficient, but the GNN benefits much more significantly. This suggests the model is able to reason that a high clustering coefficient indicates a node is more likely to be in the same community as a neighbour.

6.2 Benchmarking datasets

6.2.1 Community detection

We evaluate the GNN model on the Cora and CiteSeer datasets. For both, we use a 3-layer graph convolutional network (GCN) with hidden dimension

Table 6.13: Selected summary statistics for EN graphs as distribution varies.

$\mathbb{E}(d_{init})$	$\mathbb{E}(d_{motif})$	transitivity	diameter
1.0	1.0	.069	23.3
4.0	1.0	.064	15.6
7.0	1.0	.058	11.0
1.0	4.0	.450	17.8
4.0	4.0	.404	10.6
7.0	4.0	.359	7.6
1.0	7.0	.776	14.8
4.0	7.0	.734	8.4
7.0	7.0	.678	6.0

Table 6.14: Prediction regularisation on EN graphs.

$\mathbb{E}(d_{init})$	$\mathbb{E}(d_{motif})$	GNN-spectral	GNN-local	MLP-spectral
1.0	1.0	.576	.427	.563
4.0	1.0	.541	.410	.532
7.0	1.0	.452	.352	.436
1.0	4.0	.721	.577	.666
4.0	4.0	.764	.596	.661
7.0	4.0	.689	.520	.620
1.0	7.0	.763	.571	.684
4.0	7.0	.808	.596	.657
7.0	7.0	.806	.564	.701

Table 6.15: Local features comparison for EN graphs.

	GNN	MLP
baseline	.541	.303
with dist. dependant	.476	.323
Dropping:		
degree centrality	.536	.286
cluster coeff.	.528	.287
cluster coeff. sq.	.515	.276
avg. neighbour deg.	.567	.275
Using only:		
degree centrality	.263	.178
cluster coeff.	.429	.216
cluster coeff. sq.	.483	.244
avg. neighbour deg.	.380	.235
closeness centrality	.409	.247
eccentricity	.309	.262
is centre	.324	.132
is periphery	.288	.136

of 256 and a dropout rate of 0.1. We randomly assign nodes to training, validation and test datasets with the ratio 1:1:2. We use 2-fold cross validation to guarantee that every node is tested, and repeat the experiment 3 times, making for 6 runs in total. All models are trained for 200 epochs with early termination after 50 if no improvement is detected. Mean accuracies are reported in Table 6.16.

We begin by establishing a baseline for the GNN with the node attributes but no other derived features, (GNN+attrs). We then add the spectral features configured with 4 splits, 8 eigenvectors, and no eigenvalues or assignment variable, (GNN+attrs+4_splits). These are obtained from sung $L^{\tau-sym}$, the *sign* split method and a single virtual node with summed edge weights. The addition of these features does not lead to any increase in classification accuracy.

Next we test to see how well the community structure can be uncovered when the node attributes are dropped. We try using only the local features from Section 6.1, (GNN+locals), with a single split using 8 eigenvectors, (GNN+1_split), and then using 12 splits with 8 eigenvectors and the assignment variables for each partition. Finally for comparison, we repeat the same configurations using an MLP architecture with three hidden layers of dimension 256, 128 and 64.

For the Cora dataset, although the GNN with spectral features does not outperform the baseline, we see that in lieu of the node attributes, the spectral features alone come remarkably close to reaching the same level of performance. We observe reasonable accuracy even with a single split, but adding the features from the 12 splits produces another step up in accuracy. The results for CiteSeer are less persuasive, but we note that the MLP+attrs model gets closer to the GNN+attrs performance for CiteSeer. This indicates that the node attributes are more important for CiteSeer than for Cora, and explains the inability for the structure-only approaches to succeed as well as with Cora.

Table 6.16: Real world community detection benchmarks.

	Cora	CiteSeer
GNN+attrs	**.848**	.724
GNN+attrs+4_splits	.844	**.727**
GNN+1_split	.755	.516
GNN+12_splits	**.779**	.540

Table 6.17: ENZYMES classification accuracy

GNN+attrs	GNN+attrs+4_splits	GNN+4_splits	GNN-edge+attrs
.667	.486	.223	.623

6.2.2 Graph classification

We perform graph classification on the ENZYMES dataset using a 3-layer GNN with hidden layers with dimension 128. We use mean pooling to aggregate the node information at the final layer before passing the resultant vector to an MLP classifier with hidden dimensions 64 and 32. Dropout for all layers is set to 0.1, and models were trained for 100 epochs. Table 6.17 shows the results of 10-fold cross validation with 20% of the training data reserved for validation. We use the same configuration of models as in Section 6.2.1.

Addition of the spectral features resulted in a significant drop in performance. This was caused by the model severely overfitting to the additional features, which was not able to be sufficiently resolved by adjusting any of the regularisation parameters. We also test the edge-based clustering GNN outlined in Section 4.4, (GNN-edge+attrs). While it performs better than the GNN with spectral features, it is still markedly worse than the baseline.

Chapter 7

Discussion

We targeted the Cora dataset specifically due to the likelihood of it possessing multi-scale properties. This is highlighted by Lipov and Liò who demonstrate some performance gains using a multi-scale GCN [61]. The fact that we were not able to improve on the baseline by adding eigenfeatures serves primarily to indicate that the underlying community structure is sufficiently captured by the node attributes and citation links. More impressively however, the ability to recover the majority of the community information without *any* node attributes underlines the potential for spectral features to assist GNN architectures. This can be leveraged especially well in domains where node attributes may be noisy, corrupt, or missing entirely.

The same level of community recovery in the absence of node attributes did not occur in the CiteSeer dataset. We show in Figure 7.1 plots of the clustering coefficient against degree distribution, and can observe that both datasets appear to be hierarchical in nature due to the inverse scaling, as explained in Section 5.1. It is therefore unclear why there exists such a stark performance difference when attributes are dropped, and more work is required to understand the differences between the data.

For the graph classification tasks, we note that the edge-based clustering was not as successful as hoped. This may be due to the relatively small size of

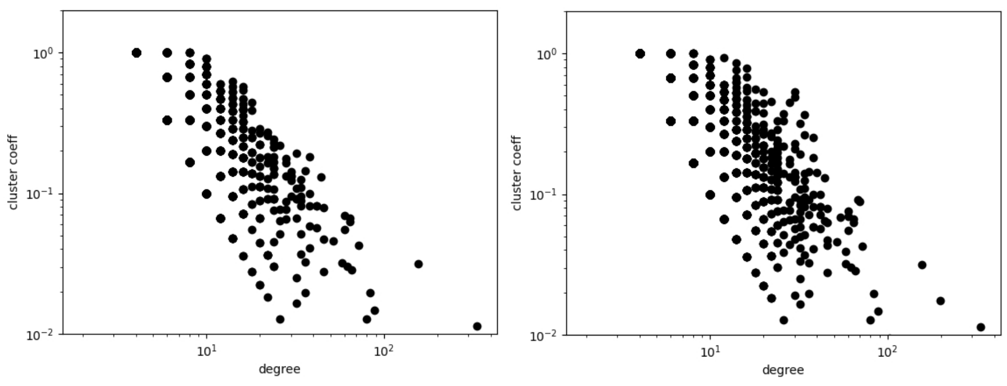

Figure 7.1: Clustering coefficient plotted against degree distribution for the Cora (left) and CiteSeer (right) datasets.

graphs in the dataset, which are less able to benefit from accelerating the flow of information. A more innovative approach to this problem is taken be Beaini *et al.* , who use spectral features to impose a vector field on a graph, instead of trying to learn this indirectly [62].

An alternative approach to modelling graphs is taken by Estrada and Ross [63]. Traditionally, protein-protein interactions are modelled as binary relationships, but this fails to capture the nuance of many more complex interactions between multiple proteins, beyond that available by analysing each of the pairwise interactions together. They therefore present a way to model PPIs as Simplicial Complexes (SCs), a generalisation of networks that define *k-simplexes* as a tuple of $(k + 1)$ nodes. Hence, a node is represented as a 0-simplex, and edges as 1-simplices, but relationships of higher degree are easily represented. They lay out a framework for defining measures of centrality in these SCs, which are then used to identify *essential proteins* – those which most seriously affect the functioning of other proteins if they themselves are disrupted. A key finding is the relatively small intersection between the essential proteins identified using 0-simplex (node) centrality and those found with 2-simplex (tripartite) centrality, (edge centrality was found to be less useful). This suggests that there may exist a significant amount of information accessible through alternate graph representations, and an interesting avenue of research could be to approach community detection from this perspective.

Another opportunity for future work may lie in adapting the chosen eigenvectors for different community sizes, instead of simply always choosing them in order of their eigenvalues' magnitude. A key observation from the single pass experiments of Section 6.1.4 was that smaller communities may resonate better with higher frequency eigenvectors than the low-frequency. A similar concept motivates Chen's *et al.* work with convolutional filters for image recognition [64], and there is justification in investigating these ideas with regards to community detection in networks. The single pass experiments demonstrated that there are certain types of data that benefit more than others from partitioning the input graph. The original motivation for our recursive partitioning approach was based on the assumption made in Section 4.1.1 that higher frequency eigenvectors would not convey enough useful information for community detection. We conclude from these experiments that this assumption is not always correct, and in fact for graphs with multi-scale properties this may also be a promising approach to incorporate.

Chapter 8

Conclusion

In this work, we explored the intersection between spectral and spatial approaches to community detection. We designed an algorithm that recursively partitions an input graph, then gathers structural features for each node through spectral analysis of each of the resulting subgraphs. These features are supplied to a neural network to predict node-cluster assignments, and we carried out a comprehensive evaluation using synthetic data designed to highlight particular qualities of graphs. Included in these are hierarchical networks, which have multi-scale properties that we felt are especially suited to our recursive algorithm.

Our evaluation included an analysis of several optimisations. We investigated different methods for choosing how we partition graphs, as well as whether to use ‘virtual nodes’ to improve connectivity of resultant subgraphs. Also explored were the use of variations on the Laplacian, including normalised and degree-regularised Laplacians. The benefits of our recursive approach were found to be successful in identifying communities in synthetic datasets, particularly for hierarchical networks. We further investigated the use of an edge-clustering GNN model for accelerating information flow for graph classification tasks, however this did not prove fruitful.

We used the Cora and CiteSeer datasets to assess the performance of our approach to community detection. The addition of structural features to the node attributes did not convey any immediate benefit to classification accuracy; however, in the absence of the attributes, the structural features alone came impressively close to matching the baseline performance.

Bibliography

[1] V. D. Blondel, J.-L. Guillaume, R. Lambiotte, and E. Lefebvre, “Fast unfolding of communities in large networks,” *Journal of statistical mechanics: theory and experiment*, vol. 2008, no. 10, P10008, 2008.

[2] M. E. Newman, “Fast algorithm for detecting community structure in networks,” *Physical review E*, vol. 69, no. 6, p. 066 133, 2004.

[3] J. Leskovec, K. J. Lang, A. Dasgupta, and M. W. Mahoney, “Community structure in large networks: Natural cluster sizes and the absence of large well-defined clusters,” *Internet Mathematics*, vol. 6, no. 1, pp. 29–123, 2009.

[4] M. Chen, T. Nguyen, and B. K. Szymanski, “A new metric for quality of network community structure,” *arXiv preprint arXiv:1507.04308*, 2015.

[5] Y.-Y. Ahn, J. P. Bagrow, and S. Lehmann, “Link communities reveal multiscale complexity in networks,” *nature*, vol. 466, no. 7307, pp. 761–764, 2010.

[6] J. J. McAuley and J. Leskovec, “Learning to discover social circles in ego networks.,” in *NIPS*, Citeseer, vol. 2012, 2012, pp. 548–56.

[7] J. Gilmer, S. S. Schoenholz, P. F. Riley, O. Vinyals, and G. E. Dahl, “Neural message passing for quantum chemistry,” in *International Conference on Machine Learning*, PMLR, 2017, pp. 1263–1272.

[8] O. Vinyals, S. Bengio, and M. Kudlur, “Order matters: Sequence to sequence for sets,” *arXiv preprint arXiv:1511.06391*, 2015.

[9] M. Zhang, Z. Cui, M. Neumann, and Y. Chen, “An end-to-end deep learning architecture for graph classification,” in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 32, 2018.

[10] R. Ying, J. You, C. Morris, X. Ren, W. L. Hamilton, and J. Leskovec, “Hierarchical graph representation learning with differentiable pooling,” *arXiv preprint arXiv:1806.08804*, 2018.

[11] Y. Ma, S. Wang, C. C. Aggarwal, and J. Tang, “Graph convolutional networks with eigenpooling,” in *Proceedings of the 25th ACM SIGKDD*

International Conference on Knowledge Discovery & Data Mining, 2019, pp. 723–731.

[12] E. Noutahi, D. Beaini, J. Horwood, S. Giguère, and P. Tossou, "Towards interpretable sparse graph representation learning with laplacian pooling," *arXiv preprint arXiv:1905.11577*, 2019.

[13] W. Jin, R. Barzilay, and T. Jaakkola, "Hierarchical generation of molecular graphs using structural motifs," in *International Conference on Machine Learning*, PMLR, 2020, pp. 4839–4848.

[14] Z. Chen, L. Chen, S. Villar, and J. Bruna, "Can graph neural networks count substructures?" In *Advances in Neural Information Processing Systems*, H. Larochelle, M. Ranzato, R. Hadsell, M. F. Balcan, and H. Lin, Eds., vol. 33, Curran Associates, Inc., 2020, pp. 10 383–10 395. [Online]. Available: `https://proceedings.neurips.cc/paper/2020/file/75877cb75154206c4e65e76b88a12712-Paper.pdf`.

[15] K. Xu, W. Hu, J. Leskovec, and S. Jegelka, "How powerful are graph neural networks?" In *International Conference on Learning Representations*, 2019. [Online]. Available: `https://openreview.net/forum?id=ryGs6iA5Km`.

[16] J. Bruna and X. Li, "Community detection with graph neural networks," 2018. arXiv: `1705.08415v2 [stat.ML]`.

[17] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.

[18] F. Lin and W. W. Cohen, "Power iteration clustering," in *ICML*, 2010.

[19] J. Bruna, W. Zaremba, A. Szlam, and Y. LeCun, "Spectral networks and deep locally connected networks on graphs," in *2nd International Conference on Learning Representations, ICLR*, vol. 2014, 2014.

[20] S. Guattery and G. L. Miller, "On the quality of spectral separators," *SIAM Journal on Matrix Analysis and Applications*, vol. 19, no. 3, pp. 701–719, 1998.

[21] I. Koutis and H. Le, "Spectral modification of graphs for improved spectral clustering," in *Advances in Neural Information Processing Systems*, H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, Eds., vol. 32, Curran Associates, Inc., 2019. [Online]. Available: `https://proceedings.neurips.cc/paper/2019/file/dbbf603ff0e99629dda5d75b6f75f966-Paper.pdf`.

[22] C. Ruiz, M. Zitnik, and J. Leskovec, "Identification of disease treatment mechanisms through the multiscale interactome," *Nature Communications*, vol. 12, no. 1, pp. 1–15, 2021.

[23] Y. Shavit, B. J. Walker, and P. Liò, "Hierarchical block matrices as efficient representations of chromosome topologies and their applica-

tion for 3C data integration," *Bioinformatics*, vol. 32, no. 8, pp. 1121–1129, Dec. 2015. [Online]. Available: `https://doi.org/10.1093/bioinformatics/btv736`.

[24] J. Shi and J. Malik, "Normalized cuts and image segmentation," *IEEE Transactions on pattern analysis and machine intelligence*, vol. 22, no. 8, pp. 888–905, 2000.

[25] A. Pothen, H. D. Simon, and K.-P. Liou, "Partitioning sparse matrices with eigenvectors of graphs," *SIAM journal on matrix analysis and applications*, vol. 11, no. 3, pp. 430–452, 1990.

[26] G. H. Golub and C. F. Van Loan, *Matrix computations.* JHU press, 2013, vol. 3.

[27] M. Fiedler, "A property of eigenvectors of nonnegative symmetric matrices and its application to graph theory," *Czechoslovak Mathematical Journal*, vol. 25, no. 4, pp. 619–633, 1975.

[28] A. Marsden, "Eigenvalues of the laplacian and their relationship to the connectedness of a graph," *University of Chicago, REU*, 2013.

[29] W. L. Hamilton, "Graph representation learning," *Synthesis Lectures on Artificial Intelligence and Machine Learning*, vol. 14, no. 3, pp. 1–159,

[30] U. Von Luxburg, "A tutorial on spectral clustering," *Statistics and computing*, vol. 17, no. 4, pp. 395–416, 2007.

[31] A. A. Amini, A. Chen, P. J. Bickel, E. Levina, *et al.*, "Pseudo-likelihood methods for community detection in large sparse networks," *Annals of Statistics*, vol. 41, no. 4, pp. 2097–2122, 2013.

[32] K. Chaudhuri, F. Chung, and A. Tsiatas, "Spectral clustering of graphs with general degrees in the extended planted partition model," in *Conference on Learning Theory*, JMLR Workshop and Conference Proceedings, 2012, pp. 35–1.

[33] M. Mihail and C. Papadimitriou, "On the eigenvalue power law," in *International Workshop on Randomization and Approximation Techniques in Computer Science*, Springer, 2002, pp. 254–262.

[34] T. Qin and K. Rohe, "Regularized spectral clustering under the degree-corrected stochastic blockmodel," *arXiv preprint arXiv:1309.4111*, 2013.

[35] V. Y. Pan and Z. Q. Chen, "The complexity of the matrix eigenproblem," in *Proceedings of the Thirty-First Annual ACM Symposium on Theory of Computing*, ser. STOC '99, Atlanta, Georgia, USA: Association for Computing Machinery, 1999, 507–516. [Online]. Available: `https://doi.org/10.1145/301250.301389`.

[36] M. Garland, "Sparse matrix computations on manycore GPUs," ser. DAC '08, Anaheim, California: Association for Computing Machinery, 2008,

2–6. [Online]. Available: `https://doi.org/10.1145/1391469.1391473`.

[37] R. B. Lehoucq, D. C. Sorensen, and C. Yang, *ARPACK users' guide: solution of large-scale eigenvalue problems with implicitly restarted Arnoldi methods*. SIAM, 1998.

[38] W. Hackbusch, "A sparse matrix arithmetic based on $\mathcal{H}$-matrices. Part I: Introduction to $\mathcal{H}$-matrices," *Computing*, vol. 62, no. 2, pp. 89–108, 1999.

[39] R. Lehoucq and D. Sorensen, "Implicitly restarted lanczos method," in *Templates for the solution of algebraic eigenvalue problems: a practical guide*, Z. Bai, J. Demmel, J. Dongarra, A. Ruhe, and H. van der Vorst, Eds., SIAM, 2000.

[40] J. Cheeger, "A lower bound for the smallest eigenvalue of the laplacian," in *Problems in Analysis*, R. C. Gunning, Ed. Princeton University Press, 2015, pp. 195–200. [Online]. Available: `https://doi.org/10.1515/9781400869312-013`.

[41] F. Chung, "Laplacians and the cheeger inequality for directed graphs," *Annals of Combinatorics*, vol. 9, no. 1, pp. 1–19, 2005.

[42] M. J. Lopez and S. S. Mohiuddin, "Biochemistry, essential amino acids," *StatPearls [Internet]*, 2020.

[43] H. W. Kuhn, "The hungarian method for the assignment problem," *Naval research logistics quarterly*, vol. 2, no. 1-2, pp. 83–97, 1955.

[44] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016, `http://www.deeplearningbook.org`.

[45] P. J. Rousseeuw, "Silhouettes: A graphical aid to the interpretation and validation of cluster analysis," *Journal of Computational and Applied Mathematics*, vol. 20, pp. 53–65, 1987. [Online]. Available: `https://www.sciencedirect.com/science/article/pii/0377042787901257`.

[46] S. Hougardy, "The floyd–warshall algorithm on graphs with negative cycles," *Inf. Process. Lett.*, vol. 110, no. 8–9, 279–281, Apr. 2010. [Online]. Available: `https://doi.org/10.1016/j.ipl.2010.02.001`.

[47] A. Decelle, F. Krzakala, C. Moore, and L. Zdeborová, "Asymptotic analysis of the stochastic block model for modular networks and its algorithmic applications," *Physical Review E*, vol. 84, no. 6, p. 066 106, 2011.

[48] C. Bordenave, M. Lelarge, and L. Massoulié, "Non-backtracking spectrum of random graphs: Community detection and non-regular ramanujan graphs," in *2015 IEEE 56th Annual Symposium on Foundations of Computer Science*, IEEE, 2015, pp. 1347–1357.

[49] F. McSherry, “Spectral partitioning of random graphs,” in *Proceedings 42nd IEEE Symposium on Foundations of Computer Science*, IEEE, 2001, pp. 529–537.

[50] S. Heimlicher, M. Lelarge, and L. Massoulié, “Community detection in the labelled stochastic block model,” *arXiv preprint arXiv:1209.2910*, 2012.

[51] E. Abbe, “Community detection and stochastic block models: Recent developments,” *The Journal of Machine Learning Research*, vol. 18, no. 1, pp. 6446–6531, 2017.

[52] E. Mossel, J. Neeman, and A. Sly, “Reconstruction and estimation in the planted partition model,” *Probability Theory and Related Fields*, vol. 162, no. 3, pp. 431–461, 2015.

[53] ——, “A proof of the block model threshold conjecture,” *Combinatorica*, vol. 38, no. 3, pp. 665–708, 2018.

[54] L. Massoulié, “Community detection thresholds and the weak ramanujan property,” in *Proceedings of the forty-sixth annual ACM symposium on Theory of computing*, 2014, pp. 694–703.

[55] E. Ravasz and A.-L. Barabási, “Hierarchical organization in complex networks,” *Physical review E*, vol. 67, no. 2, p. 026 112, 2003.

[56] J. G. Foster, D. V. Foster, P. Grassberger, and M. Paczuski, “Edge direction and the structure of networks,” *Proceedings of the National Academy of Sciences*, vol. 107, no. 24, pp. 10 815–10 820, 2010.

[57] A. A. Hagberg, D. A. Schult, and P. J. Swart, “Exploring network structure, dynamics, and function using networkx,” in *Proceedings of the 7th Python in Science Conference*, G. Varoquaux, T. Vaught, and J. Millman, Eds., Pasadena, CA USA, 2008, pp. 11 –15.

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

[59] C. L. Giles, K. D. Bollacker, and S. Lawrence, “Citeseer: An automatic citation indexing system,” in *Proceedings of the third ACM conference on Digital libraries*, 1998, pp. 89–98.

[60] K. M. Borgwardt, C. S. Ong, S. Schönauer, S. Vishwanathan, A. J. Smola, and H.-P. Kriegel, “Protein function prediction via graph kernels,” *Bioinformatics*, vol. 21, no. suppl_1, pp. i47–i56, 2005.

[61] A. Lipov and P. Liò, “A multiscale graph convolutional network using hierarchical clustering,” *arXiv preprint arXiv:2006.12542*, 2020.

[62] D. Beaini, S. Passaro, V. Létourneau, W. L. Hamilton, G. Corso, and P. Liò, “Directional graph networks,” *arXiv preprint arXiv:2010.02863*, 2020.

[63] E. Estrada and G. J. Ross, "Centralities in simplicial complexes. applications to protein interaction networks," *Journal of Theoretical Biology*, vol. 438, pp. 46–60, 2018. [Online]. Available: `https://www.sciencedirect.com/science/article/pii/S0022519317305040`.

[64] Y. Chen, H. Fan, B. Xu, Z. Yan, Y. Kalantidis, M. Rohrbach, S. Yan, and J. Feng, "Drop an octave: Reducing spatial redundancy in convolutional neural networks with octave convolution," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2019, pp. 3435–3444.