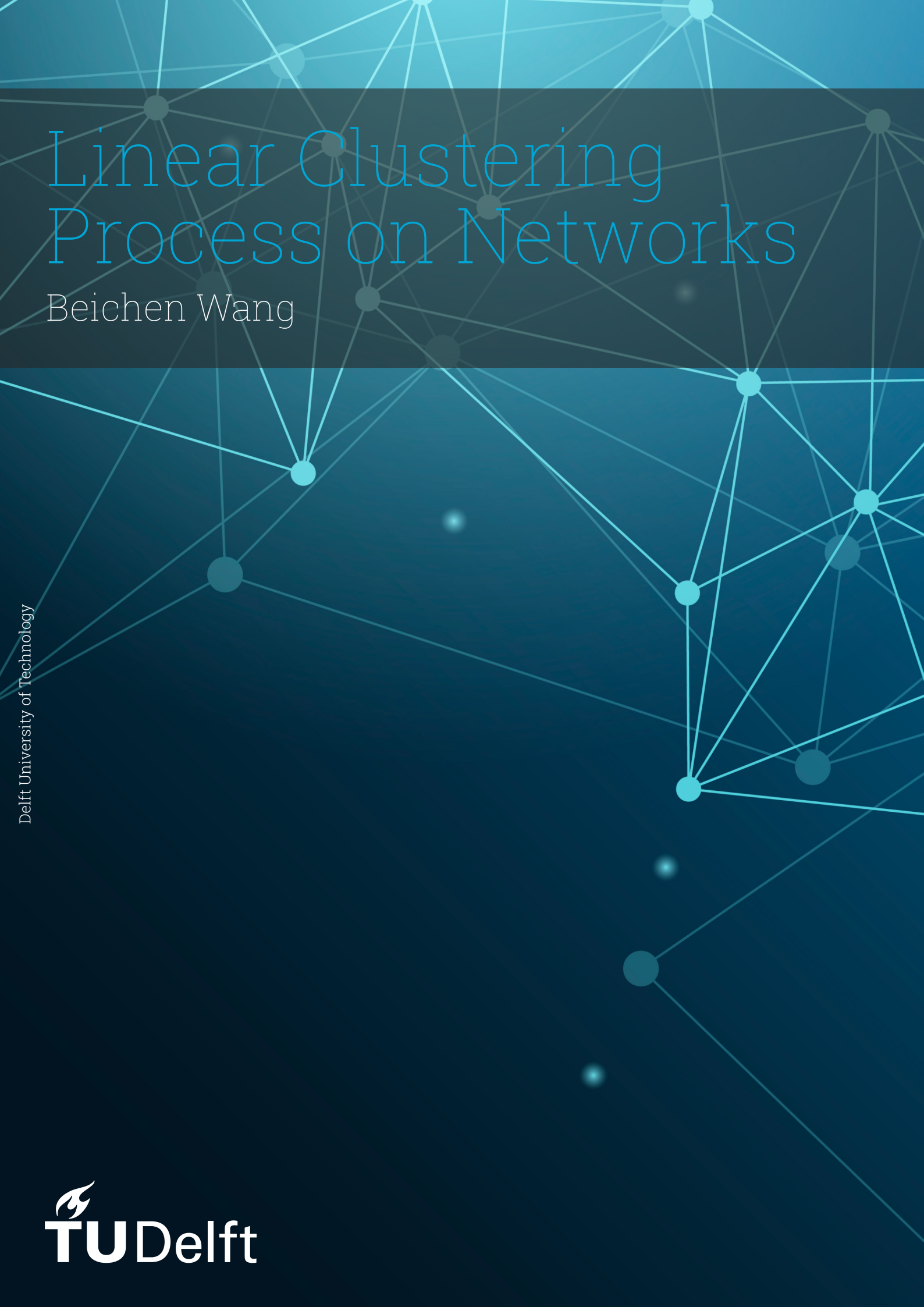

Linear Clustering Process on Networks

Beichen Wang

Delft University of Technology

TU Delft

Linear Clustering Process on Networks

by

Beichen Wang

to obtain the degree of Master of Science
in Electrical Engineering
Track Wireless Communication and Sensing
at the Delft University of Technology,
to be defended publicly on Tuesday August 29th, 2023 at 10:00 AM.

Student number:	5307171	
Project duration:	November, 2022 – August, 2023	
Thesis committee:	Prof. dr. ir. P.F.A. Van Mieghem,	TU Delft, chair
	Dr. J.L.A. (Johan) Dubbeldam,	TU Delft
	Ivan Jokić,	TU Delft
Supervisors:	Prof. dr. ir. P.F.A. Van Mieghem,	TU Delft, thesis advisor
	Ivan Jokić,	TU Delft, daily supervisor

Cover: Designed by rawpixel.com / Freepik

An electronic version of this thesis is available at http://repository.tudelft.nl/.

TU Delft

NAS

Preface

With this thesis, “Linear Clustering Process on Networks”, I complete the Master of Science degree in Electrical Engineering at Delft University of Technology. This thesis has been carried out at the Network Architectures and Services (NAS) group.

I wish to extend my deepest gratitude to my daily supervisor Ivan Jokić and thesis advisor Prof. dr. ir. P.F.A. Van Mieghem, whose unwavering guidance, insightful feedback, and dedication have been instrumental in shaping this work. Your expertise and encouragement have illuminated my path and transformed challenges into opportunities for growth.

The collaborative spirit of NAS group has provided an inspiring environment for intellectual exchange. Your discussions and perspectives during mid-term review have been invaluable in refining my ideas and methodology. Meanwhile, I would like to thank Dr. J.L.A. (Johan) Dubbeldam for being a member of my thesis committee.

In the backdrop of this academic endeavor, the steadfast support of my parents, spouse, and friends has been an unending source of motivation. Your unwavering belief in my abilities and your understanding of the sacrifices required for this pursuit have been my pillars of strength. To my beloved wife Jingxin Zhang, your patience, encouragement, and love have been my constant companions throughout this journey.

Beichen Wang
Delft, August 2023

Abstract

Community detection and graph partitioning have seamlessly integrated themselves into the fabric of network science by providing valuable insights into the structure, function, and dynamics of complex networks. In this thesis, a comprehensive performance comparison of the recently introduced Linear Clustering Process (LCP) is carried out against well-established clustering algorithms from literature. We evaluate its effectiveness using synthetic benchmarks commonly employed in the field, as well as real-world networks with both known and unknown community structures. Through our analysis, we reveal that the Linear Clustering Process consistently yields superior community partitions with optimized modularity when the clusters are well-defined compared to the majority of the assessed algorithms. Meanwhile, remarkably, this improved performance is achieved while maintaining computational complexity comparable to the simplest existing clustering algorithms. Furthermore, this thesis also provides an empirical approach for enhancing the performance of a variant of Linear Clustering Process on power-law networks.

Contents

1 Introduction

Networks pervade many disciplines, encompassing a broad range of contexts from social and information networks to biological and transportation systems [1], [2]. A network comprises the fundamental structure, determined by a graph, alongside the dynamic processes occurring within the network, as delineated by a collection of governing equations. A graph is a mathematical and conceptual representation of a collection of objects, often referred to as nodes or vertices, and the relationships or connections between these objects, represented by edges or links. Graphs can be used to model a wide range of real-world phenomena and structures and provide a way to analyze and understand the patterns of relationships between different entities. A dynamic process on a network refers to a phenomenon, activity, or change that takes place over time within the context of a graph structure. In other words, it's a process that evolves or unfolds based on interactions, relationships, or connections between nodes and links in the network.

In these complex networks, the elucidation of how entities interact and connect with each other is crucial for a comprehensive understanding of the system's overall behavior. One of the most prominent and challenging tasks in network analysis is the identification of communities, or clusters, representing groups of nodes with the majority of links connecting nodes within the same cluster, while only a few links join nodes from different clusters [3]. This task, known as community detection or graph partitioning, has been the focus of a substantial body of research, primarily due to its profound implications in various fields [3], [4].

1.1. Motivations

A reliable community detection algorithm should be able to identify "good" partitions [3]. Consequently, the implementation of a dependable quantitative criterion can be instrumental, and at times even indispensable, in distinguishing between "good" and "bad" clusterings and enhancing the algorithm's capacity to produce high-quality partitions. Newman and Girvan pioneered a quality function known as modularity [5], which subsequently inspired a series of optimization algorithms [6]. Building on this concept, Blondel *et al.* introduced the Louvain method [7], a heuristic approach based on modularity optimization. This method was lauded for its computational efficiency and was considered one of the best-performing algorithms of its time [8]. More recently, the Leiden method [9] has emerged as an enhancement to the Louvain method, designed to address some of its known limitations [9] and thereby yield higher-quality community partitions.

Spectral clustering is another widely adopted technique for community detection within a network. Essentially, this method leverages the information contained in the spectrum (eigenvalues) of the matrix representation of the network to assign nodes into clusters. Krzakala *et al.* [10] observed that the vast majority of eigenvalues from the non-backtracking matrix reside within a circle, centered at the origin of a complex plane and delineated by the square root of the largest eigenvalue as its radius. Remarkably, the quantity of eigenvalues positioned outside this circle correlates with the number of clusters. Another concept, known as the modularity eigengap [11], involves calculating the eigenvalues of the modularity matrix [6]. Each cluster is believed to correspond to a substantial eigenvalue, which markedly stands out from the remaining eigenvalues. Consequently, the number of clusters equates to the index of the

smaller eigenvalue at the point where the disparity between two adjacent eigenvalues reaches a peak, provided all eigenvalues are organized in descending order.

Recently, Shang *et al.* further contributed to this evolving field by proposing the Local Dominance algorithm [12]. This innovative approach aims to unearth the hidden hierarchy within networks by harnessing local information. Evidence indicates that the Local Dominance algorithm outperforms the Louvain method when applied to the implicated real-world networks.

Jokić and Van Mieghem proposed a novel approach [13], the Linear Clustering Process (LCP) on networks for community detection. This algorithm involves clustering by moving nodes in one-dimensional space through the use of attractive and repulsive forces. And in the same paper [13], the LCP was proven to outperform existing algorithms while maintaining similar computational complexity in a limited number of tests. Meanwhile, there are two derived variants of the proposed LCP algorithm. The first variation incorporates the actual number of clusters as an input parameter, aiming to enhance the precision and quality of community partitions. The second variation modifies the non-backtracking algorithm by substituting its original matrix with the one crafted within the LCP algorithm. This approach is designed to parallel the performance of the original non-backtracking method, providing similar efficiency and results.

1.2. Objectives and Contributions

This thesis aims to demonstrate a more extensive evaluation of LCP's performance. We firstly implement the symmetric Stochastic Block Model [14] (SBM). This model generates networks in which the clusters have the same size and the nodes have a Poisson degree distribution. Then, we further reproduce the Lancichinetti-Fortunato-Radicchi (LFR) benchmark [15] that generates networks with imbalanced communities and power-law degree distribution, which more closely resembles real-world networks. When assessing community partitions, we utilize the Element-Centric Similarity [16] (ECS) measure to provide a fairer evaluation, given the inherent bias of Normalized Mutual Information [17] [18] (NMI). Furthermore, we also equip all spectral methods with the capability to generate community partitions using K-means clustering, thus ensuring a more comprehensive and fair comparison. Despite being subjected to rigorous testing conditions, the LCP consistently showcases its superior performance. It excels in optimizing modularity, and its ability to partition communities is particularly remarkable when dealing with well-defined clusters.

Furthermore, our simulations suggest that the aforementioned second variant of the LCP algorithm exhibits performance almost equivalent to the non-backtracking method when applied to random networks. However, its efficacy drastically diminishes when operating on power-law networks. To address this limitation, we propose a dynamic adjustment strategy for the hyperparameter, determined by the power-law exponent of the degree distribution of the network. This adaptation significantly enhances the performance of the LCP variant, aligning it closely with the performance of the non-backtracking method when applied to power-law networks.

1.3. Thesis Outline

This thesis is organized as follows:

- In Chapter 2, we first provide a concise overview of fundamental graph theory and clustering concepts. And then we introduce the performance metrics used in this thesis, which include modularity, NMI, and ECS, along with various synthetic benchmarks, including the SBM and LFR benchmark.
- In Chapter 3, we offer a comprehensive overview of the LCP algorithm and two derived variants of LCP are introduced.
- In Chapter 4, we give succinct descriptions of all other clustering algorithms considered, including modularity-based methods such as the Louvain, Leiden, and Newman methods; spetral methods such as non-backtracking and modularity eigengap; and a hierarchical method called Local Dominance.
- In Chapter 5, we perform a comparative analysis, first showing the computational complexity of all algorithms and then comparing the performance of the LCP with all other algorithms considered on several synthetic benchmarks and real-world networks.

- In Chapter 6, we wrap up the paper, summarizing our findings and their implications, as well as discussing possible future work.

2

Graph or Network Clustering

A network or a graph $G(\mathcal{N}, \mathcal{L})$ contains a set of nodes $\mathcal{N}$ and a set of links $\mathcal{L}$, where the number of nodes $N = |\mathcal{N}|$ and the number of links $L = |\mathcal{L}|$. A graph can be defined by an $N \times N$ adjacency matrix A. The nodes of the graph are depicted by the rows and columns of the adjacency matrix. If a link exists between the node i and node j, then the entry in the matrix $a_{ij} = 1$; otherwise, the entry $a_{ij} = 0$. Then, the $N \times 1$ degree vector d can be calculated as $d = A \cdot u$, where u is the $N \times 1$ all-one vector. And the $N \times N$ degree diagonal matrix Δ is defined as $\Delta = \text{diag}(d)$.

The set of neighboring nodes of node i is denoted by $\mathcal{N}_i = \{k \mid a_{ik} = 1, k \in \mathcal{N}\}$. So the degree of node i can also be derived by the cardinal number of this set, say $d_i = |\mathcal{N}_i|$. $\mathcal{N}_i \cap \mathcal{N}_j$ denotes the set of common neighbors of node i and node j, while the set of neighbors that are not shared by these two nodes is $\mathcal{N}_i \setminus \mathcal{N}_j$. Therefore, the degree of node i is also equivalent to the summation of the number of shared and distinct neighbors between nodes i and j:

$$d_i = |\mathcal{N}_i \cap \mathcal{N}_j| + |\mathcal{N}_i \setminus \mathcal{N}_j|. \tag{2.1}$$

2.1. Clustering Performance

2.1.1. Modularity

Newman and Girvan [5] introduced the concept of modularity to facilitate network partitioning. The modularity measure, denoted as m in equation (2.2), quantifies the difference between the actual number of links between nodes belonging to the same community and the expected number of such links in a randomly connected network. The equation is given by:

$$m = \frac{1}{2L} \cdot \sum_{i=1}^{N} \sum_{j=1}^{N} \left(a_{ij} - \frac{d_i \cdot d_j}{2L} \right) \cdot \mathbf{1}_{\{i \text{ and } j \in \text{ same cluster}\}}, \tag{2.2}$$

where $\mathbf{1}_x$ is an indicator function that equals 1 if statement x is true, and 0 otherwise. The modularity value, m, approaching 0 indicates that the estimated partition is as good as a random one. Conversely, a modularity value close to 1 suggests a clear partitioning of the network into distinct clusters. It is important to note that optimizing modularity is NP-complete [19], and approximations have been proposed [20]. By defining the $N \times N$ cluster matrix C as:

$$C_{ij} = \begin{cases} 1 & \text{if nodes } i \text{ and } j \text{ belong to the same cluster} \\ 0 & \text{otherwise,} \end{cases} \tag{2.3}$$

the equation (2.2) can also be rewritten as a quadratic form:

$$m = \frac{1}{2L} \cdot u^T \cdot \left(A \circ C - \frac{1}{2L} \cdot \left(d \cdot d^T \right) \circ C \right) \cdot u, \tag{2.4}$$

where the symbol $\circ$ represents the Hadamard product [21]. The number of clusters in the network is denoted as c and the $c \times 1$ vector $n = \begin{bmatrix} n_1 & n_2 & \dots & n_c \end{bmatrix}$, with n_i representing the number of nodes in cluster i, specifies the size of each cluster.

2.1.2. Normalized Mutual Information

Danon *et al.* [17] introduced the normalized mutual information (NMI) metric as a means of comparing partitions in network analysis. The metric relies on a confusion matrix, denoted as F, which represents the correspondence between the original communities and the estimated clusters. The ij-th element F_{ij} in the confusion matrix indicates the number of nodes belonging to both the true community i and the estimated community j. The normalized mutual information metric, denoted as $I_n(P_0, P_e)$, between the known partition P_0 and the estimated partition P_e is defined in [17] as:

$$I_n(P_0, P_e) = \frac{-2\sum_{i=1}^{c_0}\sum_{j=1}^{c_e} F_{ij}\log\left(\frac{F_{ij}N}{F_{i.}F_{.j}}\right)}{\sum_{i=1}^{c_0} F_{i.}\log\left(\frac{F_{i.}}{N}\right) + \sum_{j=1}^{c_e} F_{.j}\log\left(\frac{F_{.j}}{N}\right)}, \tag{2.5}$$

where c_0 represents the number of known clusters, c_e represents the number of estimated clusters, $F_{i.}$ denotes the sum of the i-th row in F, and $F_{.j}$ denotes the sum of the j-th column. The NMI metric takes a value of 1 when two partitions are identical, and tends towards 0 when the partitions are independent. The NMI measure has been widely employed in evaluating the performance of various clustering algorithms [3] and continues to be a valuable tool in network analysis.

2.1.3. Element-centric Similarity

Gates *et al.* [16] introduced the Element-centric Similarity (ECS) metric to tackle the biases inherent in existing clustering comparison measures. The authors highlight that such biases are pervasive in popular metrics. For instance, when comparing the similarity between two clusterings, one is typically fixed as the ground truth while the number of clusters in the other is increased. As a consequence, the NMI value rises, whereas the ECS value declines. This indicates that NMI exhibits a bias towards a greater number of clusters, whereas ECS remains unbiased in this regard.

The process of ECS commences by computing the "personalized PageRank" or "random walk with restart" p_{ij}, incorporating all possible paths between elements to derive the equilibrium distribution for a personalized diffusion process on the graph:

$$p_{ij} = (\alpha_{ecs}/\left|c_\beta\right|) + (1-\delta_{ij})(1-\alpha_{ecs}))\delta_{\beta\gamma}, \tag{2.6}$$

where δ is the Kronecker delta function, element v_i is in cluster c_γ, and element v_j is in cluster c_β. And $|c_\beta|$ denotes the cluster size of c_β. $1.0-\alpha_{ecs}$ represents the restart probability, where $\alpha_{ecs}=0.9$.

With the personalized PageRank p_{ij}, we can further calculate the element-wise similarity of an element v_i in two clusterings $\mathcal{A}$ and $\mathcal{B}$,

$$S_i(\mathcal{A},\mathcal{B}) = 1.0 - \frac{1}{2\alpha_{ecs}}\sum_{j=1}^{N}\left|p_{ij}^{\mathcal{A}} - p_{ij}^{\mathcal{B}}\right|. \tag{2.7}$$

Finally, the ECS score $S(\mathcal{A},\mathcal{B})$ of two clusterings $\mathcal{A}$ and $\mathcal{B}$ is the average of $S_i(\mathcal{A},\mathcal{B})$,

$$S(\mathcal{A},\mathcal{B}) = \frac{1}{N}\sum_{i=1}^{N} S_i(\mathcal{A},\mathcal{B}). \tag{2.8}$$

The closer the value of ECS is to 1, the more similar the two clusterings are, while the closer the value of ECS is to 0, the more dissimilar the two clusterings are.

Furthermore, it is noteworthy that the complete version of ECS possesses the capability to assess both overlapping clustering and hierarchical clustering. However, within the scope of this thesis, all utilized algorithms, benchmarks, and real-world networks exclusively pertain to classic community problem: where each node is singularly allocated to a lone community. As such, our reimplementation of ECS focuses solely on this facet of its functionality.

2.2. Benchmarks

2.2.1. Stochastic Block Model

The clustering methods employed in this thesis will be firstly benchmarked using random graphs generated by the Stochastic Block Model (SBM), which was proposed by Holland [14]. The SBM generates

a random graph with a community structure, where the presence of a link between two nodes depends on whether they belong to the same cluster or not, and the probability of the link varies accordingly. This thesis specifically focuses on the symmetric stochastic block model (SSBM), which involves defining only two distinct probabilities for link existence. If two nodes belong to the same cluster, they are connected by a link with a probability of p_{in}. Otherwise, a direct link exists between them with a probability of p_{out}. Communities are formed when the link density within clusters is greater than the inter-community link probability, i.e., when $p_{in} > p_{out}$. Additionally, the clusters are constrained to have the same size, denoted by $n_i = N/c$, where $i \in 1, 2, \ldots, c$. This constraint ensures that the expected degree is equal for all nodes, regardless of their cluster membership. The expected degree $E[D]$ is given by the equation:

$$E[D] = \frac{b_{in} + (c-1) \cdot b_{out}}{c}. \tag{2.9}$$

In this thesis, we consider a sparse and assortative variant of the SSBM. The terms sparse and assortative imply that the link probabilities, $p_{in} = b_{in}/N$ and $p_{out} = b_{out}/N$, are defined based on positive constants b_{in} and b_{out}, which remain fixed as the network size N approaches infinity. Decelle *et al.* [22, 23] discovered that when the difference between b_{in} and b_{out} surpasses a detectability threshold, represented by the equation:

$$b_{in} - b_{out} > c \cdot \sqrt{E[D]}, \tag{2.10}$$

it becomes theoretically possible to accurately identify the cluster membership of nodes. Conversely, if the difference between b_{in} and b_{out} falls below this threshold, the network's community structure cannot be distinguished from randomness. This threshold (2.10) represents a critical transition point between the undetectable and theoretically detectable regimes of the SSBM.

2.2.2. LFR Benchmark

Lancichinetti *et al.* [15] proposed the LFR benchmark as an alternative to SSBM graphs, aiming to generate more realistic random graphs that incorporate inherent community structures. Unlike SSBM graphs, where all nodes have the same expected degree, the authors argue that real-world networks often exhibit heterogeneous degree distributions. Moreover, the tails of these distributions are frequently characterized by power laws [24]. In contrast, the LFR benchmark takes into account the observed properties of real-world networks, where community size distributions often follow heavy-tailed distributions [25]. This benchmark produces graphs with the following characteristics:

- Each node's degree is sampled from a power law distribution with an exponent determined by the input parameter γ;
- The size of each community is sampled from a power law distribution with an exponent determined by the input parameter β_{lfr};
- A fraction $1 - \mu$ of the links of each node are assigned as intra-community links.

In addition to the parameters mentioned above, the LFR benchmark requires inputs for the network size N, the average degree d_{av}, and the number of communities c.

2.3. Random Network Models

2.3.1. Erdős-Rényi Model

The Erdős-Rényi (ER) model [26] is a fundamental model in the field of network theory. There are two variants of this model: the $G(N, L)$ model and the $G(N, p_{ER})$ model.

For the $G(N, L)$ model, a graph is generated by choosing at random from all possible graphs consisting of N nodes and L links. Essentially, this model begins with N isolated nodes and proceeds to add L links at random, ensuring the avoidance of duplicating links between the same pair of nodes. The $G(N, p_{ER})$ model builds a graph with N nodes, where the probability p_{ER} determines whether each pair of nodes is connected by an link. The decision to create an link between each pair of nodes is taken independently. In our simulations, we implement the $G(N, p_{ER})$ model.

It should be noted that the Erdős-Rényi model falls short in replicating all features of real-world networks. For instance, many real-world networks exhibit a higher clustering coefficient, which suggests a pronounced tendency for nodes within the graph to form clusters, and a power-law degree distribution.

These are properties that are not typically observed in Erdős-Rényi networks [27]. To better represent these characteristics, more complex models, such as the Barabási-Albert (BA) model, have been developed.

2.3.2. Barabási-Albert Model

The Barabási-Albert (BA) model [28] is a model of network growth. This model specifically aims to generate random scale-free networks, which are characterized by node degrees (the number of links incident to a node) that follow a power-law distribution. A key feature of scale-free networks is that there are a few nodes that have a significantly higher number of connections compared to others, often referred to as ”hubs”.

The main characteristic of Barabási-Albert model is called preferential attachment, which means that new nodes are more likely to connect to nodes that already have many connections. This ”rich-get-richer” phenomenon leads to the power-law degree distribution that characterizes scale-free networks.

Here is a detailed description of the BA model's steps:

- Begin with a small number m_0 of nodes, usually $m_0 > 1$, which are connected in some arbitrary manner;
- At each time step, add a new node with $m_{BA} \leq m_0$ links that connect the new node to m_{BA} existing nodes;
- The probability p_i that the new node will connect to node i depends on the degree d_i of node i, such that $p_i = d_i / \sum_j d_j$. That is, the more connections node i has, the more likely it is to receive more connections.

These steps are repeated until the network reaches the desired size. The resulting network has a scale-free degree distribution, meaning that the fraction $P(d)$ of nodes in the network having d connections to other nodes goes for large values of d as $P(d) \propto d^{-\gamma}$, where γ is a constant whose value typically lies in the range $2 < \gamma < 3$.

While the Barabási-Albert model succeeds in explaining the presence of hubs and the power-law degree distribution in many real-world networks, Barabási-Albert model does not capture all aspects of real-world network structure. For example, it does not account for the high clustering coefficient seen in many real-world networks. Networks possessing such a property are accurately represented by the Watts-Strogatz model.

2.3.3. Watts-Strogatz Model

The Watts-Strogatz (WS) model [29] is a stochastic graph generation model that yields graphs characterized by small-world properties, encompassing short average path lengths and high clustering. It was designed to mimic the properties of many real-world networks, which are neither completely regular nor completely random.

The Watts-Strogatz model is based on the idea of ”rewiring” a regular lattice. Here are the steps of constructing a Watts-Strogatz network:

- Start with a regular lattice: A graph of N nodes is created where each node is connected to its k_{WS} nearest neighbors ($k_{WS}/2$ on each side, assuming periodic boundary conditions which makes the lattice a ring);
- Rewiring process: For every link in the network, rewire the target node with probability p_{WS}. Rewiring works as follows: for a selected link, change one of its nodes (keeping the source node) to a node chosen uniformly at random over the entire ring, with duplicate links disallowed.

The rewiring probability p_{WS} is a key parameter of the Watts-Strogatz model. Depending on the parameter p_{WS} value, Watts-Strogatz model generates different types of graph topologies:

- When $p_{WS} = 0$, the model starts as a regular graph with high clustering and large characteristic path length;
- When $p_{WS} = 1$, the model becomes a random graph, similar to an Erdős-Rényi graph, having a low clustering coefficient and small characteristic path length;
- For $0 < p_{WS} < 1$, the model exhibits small-world properties, with a relatively high clustering coefficient and small characteristic path length.

It is important to note that the Watts-Strogatz model does not generate scale-free networks, which is a common property of many real-world networks, as it lacks a power-law degree distribution.

3

Linear Clustering Process

In [13], Jokic and Van Mieghem proposed a linear process consisting of attractive and repulsive forces between nodes of a network, which can be utilized to estimate partitions.

3.1. Linear Clustering Process (LCP) on a Network

3.1.1. From node-level governing equation to the network level

In the graph G, every node i is allocated a position $x_i[k]$ on a linear axis, which corresponds to a one-dimensional space, at a specific, discrete point in time, denoted as k. LCP consists of two opposite and simultaneous forces that change nodal position in time:

- Attraction: Adjacent nodes that have a lot of common neighbors are drawn towards each other with a force that is proportional to the quantity of shared neighbors. In particular, the attractive force between node i and its neighboring node j is proportional to $\alpha \cdot (|\mathcal{N}_j \cap \mathcal{N}_i| + 1)$, where α is the attraction strength and $\mathcal{N}_j \cap \mathcal{N}_i$ denotes the set of common neighbors of node i and node j;
- Repulsion: Adjacent nodes are repulsed with a force proportional to the number of neighbours they do not share. In particular, the repulsive force between node i and its neighboring node j is proportional to $\delta \cdot (|\mathcal{N}_j \setminus \mathcal{N}_i| - 1)$, where δ is the repulsive strength and $\mathcal{N}_j \setminus \mathcal{N}_i$ denotes the set of neighbors of node j that do not belong to node i, accounted for the direct link between node i and node j, that is contained in the above set. To obtain symmetric repulsion force between node i and node j, when these nodes are interchanged, the repulsion force is defined to be proportional to $\delta \cdot (|\mathcal{N}_j \setminus \mathcal{N}_i| + |\mathcal{N}_i \setminus \mathcal{N}_j| - 2)$.

Therefore, the governing equation at position $x_i[k]$ of node i at discrete time k is

$$x_i[k+1] = x_i[k] + \sum_{j \in \mathcal{N}_i} \left(\frac{\alpha \cdot (|\mathcal{N}_j \cap \mathcal{N}_i| + 1)}{d_j d_i} - \frac{\frac{1}{2} \cdot \delta \cdot (|\mathcal{N}_j \setminus \mathcal{N}_i| + |\mathcal{N}_i \setminus \mathcal{N}_j| - 2)}{d_j d_i} \right) \cdot \Big(x_j[k] - x_i[k] \Big) \quad (3.1)$$

The node-level governing equation of LCP could be further transformed to the network level and rewritten to a matrix form. The discrete time process (3.1) satisfies the following linear matrix difference equation, as derived in [13, Theorem 1]

$$x[k+1] = (I + W - \mathsf{diag}\,(W \cdot u)) \cdot x[k], \quad (3.2)$$

where the $N \times 1$ all-one vector is notated as u, the $N \times N$ identity matrix is denoted by I, while the $N \times N$ topology-based matrix W is defined as

$$W = (\alpha + \delta)\, \Delta^{-1} \cdot \left(A \circ A^2 + A \right) \cdot \Delta^{-1} - \frac{1}{2} \cdot \delta \left(\Delta^{-1} \cdot A + A \cdot \Delta^{-1} \right) \quad (3.3)$$

The attractive force between two adjacent nodes is always of higher strength than the repulsive force, preserving the system's stability. The bounds of the attraction α and repulsion δ strengths are demonstrated in [13, p. 5] to preserve the stability of the LCP process. However, the dominance of attraction forces governs the LCP process eventually to a trivial steady state, where each node occupies the same position, as derived in [13, p. 4].

3.1.2. Time-dependence of the LCP

The explicit solution of the equation (3.2) is

$$x[k] = (I + W - \text{diag}\,(W \cdot u))^k\, x[0] \tag{3.4}$$

where the k-th component of the initial position vector is $(x[0])_k = k$.

The convergence of the linear discrete-time system described in (3.2) to a steady-state is achievable only when the matrix $(I + W - \text{diag}\,(W \cdot u))$ possesses eigenvalues with absolute values smaller than 1, and the largest eigenvalue is exactly 1. This implies that the requirement for achieving a steady state is the alignment of all nodes to the same position. However, when we closely examine the governing equation (3.1), the steady-state solution seems trivial. This is because the sum cancels out, and the definition of steady state states that $x[k+1] = x[k]$, a condition that holds true for any discrete-time independent vector. Therefore, the matrix equation (3.2) can be written as

$$x[k+1] - x[k] = (W - \text{diag}\,(W \cdot u)) \cdot (x[k] - u)$$

The eigenvalue decomposition of the $N \times N$ governing matrix $W - \text{diag}\,(W \cdot u)$ can be written as

$$W - \text{diag}\,(W \cdot u) = Y \text{diag}(\beta) Y^T \tag{3.5}$$

where the $N \times 1$ eigenvalue vector $\beta = (\beta_1, \beta_2, \cdots, \beta_N)$ with $\beta_1 \geq \beta_2 \geq \cdots \geq \beta_N$ and Y is the $N \times N$ orthogonal eigenvector matrix with the eigenvectors $y_1, y_2, \cdots, y_N$ in the columns obeying $Y^T Y = YY^T = I$. Since $\beta_1 = 0$ and $y_1 = u/\sqrt{N}$, it holds for $k > 1$ that $u^T y_k = 0$, which implies that the sum of the components of eigenvector y_k for $k > 1$ is zero. The position vector in (3.4) is rewritten as

$$x[k] = Y \text{diag}(1+\beta)^k Y^T x[0] = \sum_{j=1}^{N} (1+\beta_j)^k y_j \left(y_j^T x[0]\right)$$

Therefore, we arrive at

$$x[k] - \frac{u^T x[0]}{\sqrt{N}} u = \sum_{j=2}^{N} (1+\beta_j)^k (y_j^T x[0])\; y_j \tag{3.6}$$

As elucidated earlier, the left-hand side represents a translated position vector and holds no significant influence over the clustering process from a physical standpoint. Since $-1 < \beta_j < 0$ for $j > 1$, equation (3.6) demonstrates that, for $k \to \infty$, the right-hand side approaches zero, rendering the steady-state solution evidently unremarkable in the context of the clustering process. This allows us to rewrite (3.6) as

$$x[k] - \frac{u^T x[0]}{\sqrt{N}} u = (1+\beta_2)^k \left((y_2^T x[0])\; y_2 + \sum_{j=3}^{N} \left(\frac{1+\beta_j}{1+\beta_2} \right)^k (y_j^T x[0])\; y_j \right). \tag{3.7}$$

Since $\beta_1 = 0$ and $-1 < \beta_j < 0$ for $j > 1$, $|1+\beta_2| > |1+\beta_3|$ holds. Therefore, we observe that

$$\frac{x[k] - \frac{u^T x[0]}{\sqrt{N}} u}{(1+\beta_2)^k\, (y_2^T x[0])} = y_2 + O\left(\frac{1+\beta_3}{1+\beta_2} \right)^k, \tag{3.8}$$

The left-hand side of (3.8), representing a shifted or normalized position vector, gravitates toward the second eigenvector y_2, accompanied by an exponentially diminishing error as k increases. However, this only holds true for sufficiently large, but not excessively large, values of k. Thus, the information used for clustering the graph is provided by this scaled and shifted position vector.

A block diagonal structure of the $N \times N$ adjacency matrix A can be found by sorting the $N \times 1$ eigenvector y_2, in ascending or descending order. This sorting process leads to a graph relabeling, where each node is assigned a new label based on its position in the sorted eigenvector, denoted as $\hat{y_2}$. Consequently, the original two-dimensional clustering problem is transformed into a one-dimensional problem. In this new formulation, we either group nodes with similar values in $\hat{y_2}$ to form communities, or we determine the community boundaries by optimizing a quality function, such as modularity. The LCP method makes use of modularity for this purpose. And the next section describes this process in detail.

3.1.3. Community detection based on the eigenvector y_2

The dynamic interaction between attractive and repulsive forces among nodes propels the nodal positions over discrete time steps k, guiding them toward a trivial steady state $\lim_{k\to\infty} x[k] = u$. Concurrently, the scaled and shifted position vector, denoted as $x[k]$, gradually converges over time towards the second-largest eigenvector, y_2, characterized by an exponentially diminishing margin of error.

Through the sorting of eigenvectors from y_2 to $\hat{y_2}$, the y_2 components undergo reordering, thereby unveiling a block diagonal structure of the adjacency matrix A upon the nodes' consequent relabeling within the network. Therefore, a $N \times N$ permutation matrix R could be defined in a way the following equation and inequality hold:

$$\begin{aligned} \hat{y_2} &= R \cdot y_2, \\ (\hat{y_2})_i = (y_2)_{r_i} &\leq (\hat{y_2})_j = (y_2)_{r_j}, \quad i < j, \end{aligned} \tag{3.9}$$

where the $N \times 1$ ranking vector $r = R \cdot w$ and $w = [1, 2, \ldots, N]$. And r_i denotes the ranking of node i in the eigenvector y_2. Then, the $N \times N$ relabeled adjacency matrix $\hat{A}$, the $N \times 1$ relabeled degree vector $\hat{d}$ of G, and the $N \times 1$ sorted eigenvector $\hat{y_2}$ can be further defined by the permutation matrix R as follows:

$$\begin{cases} \hat{A} &= R^T \cdot A \cdot R \\ \hat{d} &= R \cdot d \\ \hat{y_2} &= R \cdot y_2. \end{cases} \tag{3.10}$$

Clusters form among groups of nodes exhibiting relatively minor discrepancies in the eigenvector y_2 components, yet substantial distinctions compared to other network nodes. Consequently, the challenge of community detection transitions into the task of identifying contiguous ranges of comparable values within the sorted eigenvector $\hat{y_2}$.

The eigenvector y_2 serves as a continuous gauge of the similarity between the neighboring nodes of two given nodes. Clusters are estimated for a node ranking r derived from sorted eigenvector y_2 through recursive optimization of the modularity m. In the first iteration, all possible partitions of the network into two clusters are examined and their modularity computed. The partition that yields the highest modularity is selected. In the second iteration, the same procedure is repeated for each subgraph, and the best partitions into two clusters are found. Once the best partitions for both sub-graphs are determined, they are adopted if the modularity of the generated partition surpasses the modularity of a parent cluster from the previous iteration. The recursive process is halted when there can be no further improvement in the modularity. The pseudo-code for this recursive algorithm is in the Appendix F of [13].

3.1.4. Weakening strength of inter-cluster forces

The metric definition of nodal positions within LCP stands as a remarkable attribute, enabling diverse clustering methodologies. The extent of positioning disparity between any two nodes, whether adjacent or not, serves as an indication of their potential cluster membership. Furthermore, this position metric facilitates the categorization of links into intra- and inter-community distinctions. The iterative linear clustering process (3.1) involves successive iterations in which inter-community link weights are identified and scaled.

The attraction and repulsive forces manifest as linear functions relative to the positional disparity between neighboring nodes. While this linearity promotes analytical rigor and simplification, it also introduces complexities. As nodes become more distantly positioned, the intensities of both attractive and repulsive forces amplify. Conversely, as neighboring nodes draw closer along a line, the forces diminish, ultimately reaching zero as nodes converge. Additionally, the attractive force between neighboring nodes consistently outweighs the repulsive force, leading the process toward a trivial steady state.

Non-linearity in forces can be progressively infused into the linear clustering process through incremental scaling of inter-community link weights across iterations. This deliberate reduction in force strength between nodes from distinct clusters effectively dampens the impact of inter-community links, based on the partition delineated in prior iterations.

In particular, the difference $|\,(y_2)_i - (y_2)_j\,|$ in eigenvector component y_2 between nodes i and j signifies the resemblance between their respective neighboring nodes. A normalized metric for gauging

dissimilarity in adjacent nodes i and j involves assessing the difference $(|r_i - r_j|)$ in their rankings within the sorted eigenvector $\hat{y_2}$. Consequently, connections linking nodes exhibiting the most significant ranking disparities are apt to be inter-community links. The $N \times N$ scaling matrix S is defined as follows:

$$s_{ij} = \begin{cases} 1, & \text{if } |r_j - r_i| < \theta_r \\ \upsilon, & \text{otherwise ,} \end{cases} \tag{3.11}$$

where the ij-th element equals 1 if the absolute value of the ranking difference between nodes i and j is lower than the threshold θ_r, otherwise is equal to a small positive value $0 \leq \upsilon \leq 1$. Based on this scaling matrix S, the governing equation in (3.2) can be updated as follows:

$$x[k+1] = \left(I + \tilde{W} - \text{diag}\left(\tilde{W} \cdot u\right)\right) \cdot x[k],$$

where $\tilde{W} = S \circ W$. The clustering process in (3.2) is solely affected by the scaling of link weights in (3.11), as stipulated in the aforementioned equation. Nonetheless, in each iteration, modularity-based community detection works with the $N \times N$ adjacency matrix A. As a result, the process is aided by the implementation of weight scaling for inter-community connections in the network to enhance the distinction between clusters (i.e. ultimately leading to improved relabeling in (3.10)). This is achieved without altering the adjacency matrix A and, consequently, without causing any adverse impact on the optimization of modularity m, which is introduced by the algorithm in the Appendix F of [13]. The quality of the identified graph partition is significantly enhanced by the scaling of link weights between clusters.

3.2. LCP for a known number of communities

The aforementioned algorithm for recursively optimizing modularity can also be applied to graph partitioning with a known number of communities c. In this scenario, the recursive procedure outlined is halted not when the modularity can no longer be improved, but at iteration $(\log_2 c+1)$. In every iteration, the partition with the highest modularity is accepted, regardless of whether it is negative.

As a consequence, $2c$ estimated clusters are obtained using the $2c \times 2c$ aggregated modularity matrix M_c:

$$(M_c)_{gh} = \sum_{i \in g, j \in h} \left(\hat{A} - \frac{1}{2L} \cdot \hat{d} \cdot \hat{d}^T\right)_{ij}, \tag{3.12}$$

where $g, h \in \{1, 2, \ldots, 2c\}$ denote estimated clusters. The capability to merge adjacent clusters is facilitated by the aggregated modularity matrix M_c, with the aim of achieving c communities in an iterative manner. The $(2c - 1 \times 1)$ vector ν is observed, where $\nu_g = (M_c)_{g,g+1}$. The merging of two adjacent clusters that minimally impacts the modularity index m is determined by identifying the maximum element of ν. By iteratively performing this process c times, the graph partition in c clusters is ultimately achieved. This variant is denoted as LCP_c.

The above method can be termed as a priori LCP_c, given that the algorithm possesses awareness of the real number of clusters from the beginning and in each iteration. Furthermore, an attempt was made with a posteriori LCP_c, wherein the original LCP remains unaltered, and only an iteration that achieves the desired clustering in the final step is selected. In cases where multiple iterations attain the target cluster number, the one exhibiting the highest modularity is chosen; if no such iterations exist, the iteration with the utmost modularity across all iterations is selected. Subsequent to testing, it was observed that this approach does not yield any discernible distinction from the original LCP across various benchmarks, leading to the decision to discard this approach.

3.3. Non-backtracking variant of LCP

Angel *et al.* [30] shows another approach of computing the eigenvalues of non-backtracking matrix, which will be explained in Section 4.2.1. Instead of directly decomposing the $2L \times 2L$ non-backtracking matrix B, these $2N$ non-trivial eigenvalues are also contained in the decomposed $2N \times 2N$ matrix B^*:

$$B^* = \begin{bmatrix} A & I - \Delta \\ I & O \end{bmatrix}, \tag{3.13}$$

where O denotes the $N \times N$ all-zero matrix. This B^* matrix can be rewritten as:

$$B^* = \begin{bmatrix} I + (A - \Delta) + (\Delta - I) & -(\Delta - I) \\ I & O \end{bmatrix} \tag{3.14}$$

Therefore, this matrix can be viewed as a state-space matrix for a process happening within a network, mirroring the LCP process we described in equation 3.2. In this process, the final N states are responsible for holding the delayed values of the initial N states. The matrix B^* establishes a set of N second-order differential equations. The governing equation that dictates the position of node i is defined by

$$x_i[k+1] = x_i[k] + \sum_{j \in \mathcal{N}_i} (x_j[k] - x_i[k]) + (d_i - 1) \cdot (x_i[k] - x_i[k-1]), \tag{3.15}$$

where the second term can be considered as a uniform attractive force between neighboring nodes, whereas in LCP, the intensity of the attractive force is dependent on the number of common neighbors between two adjacent nodes. Meanwhile, LCP also suggests a repulsive force between adjacent nodes, equation 3.15 describes node i as being pushed away from its former position $x_i[k]$ in the direction of its most recent position change $x_i[k] - x_i[k-1]$.

Therefore, corresponding to B^*, $2N \times 2N$ matrix W^* can be defined as:

$$W^* = \begin{bmatrix} I + \alpha \cdot (A \circ A^2 + A - \mathsf{diag}((A \circ A^2 + A) \cdot u) + (\Delta - I) & -(\Delta - I) \\ I & O \end{bmatrix} \tag{3.16}$$

where attractive force is retained according to but repulsive force is discarded by letting $\delta = 0$. The remaining part aligns with the principles of the non-backtracking clustering method. The number of clusters c is determined by counting the number of real eigenvalues in W^* that exceed the square root of the largest real eigenvalue.

This variant is denoted as $\mathsf{LCP_n}$ and we test it on several benchmarks, random network models and real-world networks. Specific results and analysis are presented in Chapter 5. The positive aspect is that the algorithm performs almost identically to non-backtracking on the Stochastic Block Model and random network models, which proves the success of the algorithm design. However, the algorithm performs poorly on power-law networks, such as the LFR benchmark and some of real-world networks as shown in Figure 3.1. After analysis and testing, it was found that for power-law networks, the attraction strength α has a significant impact on the performance of the algorithm. Previously we fix $\alpha = 0.95$, mainly because an α close to 1 helps maximize the difference between the second largest eigenvalue β_2 and the third largest eigenvalue β_3 of the $N \times N$ matrix $W - \mathsf{diag}(W \cdot u)$. Therefore, for the power-law network, we adopt the strategy of dynamically adjusting the value of α, which is described below.

For a power-law network, its degree distribution should conform

$$P(d) \propto \frac{1}{d^\gamma}, \tag{3.17}$$

where $P(d)$ denotes the probability that the node has degree d and γ is a positive real number. In our case, we treat networks with γ greater than 1 as power-law networks [31], because the degree distribution of some random model networks may also have positive γ values, but they are hardly greater than 1. Thus, when we have a power-law network, we can thereby calculate its γ based on its degree distribution.

Firstly take the logarithm of both sides of the equation (3.17), we have

$$\log P(d) = -\gamma \cdot \log d \tag{3.18}$$

Here we obtain a linear equation and we can then perform a linear fit using the degrees of the nodes and their corresponding probabilities. The probability density function $f_D(d)$ of the power-law distribution [32] is as follows:

$$f_D(d) = c_p \cdot d^{-\gamma}, \tag{3.19}$$

where $c_p = (\gamma - 1) \cdot d_{min}^{(\gamma-1)}$ denotes the normalization constant. For a given network, we utilize the $N \times 1$ degree vector d to estimate the corresponding power-law exponent γ, which helps determining whether a network is a power-law network and how "skew" the degree distribution is. In particular, the power-law exponent γ determines the shape of the distribution [31], [33]:

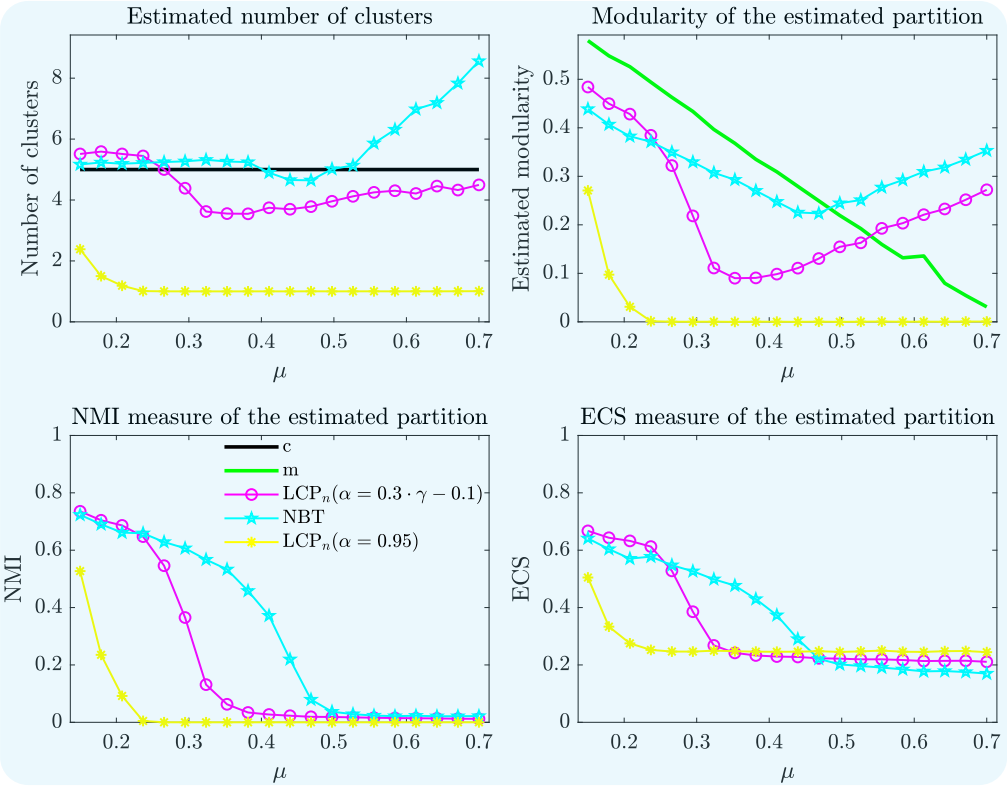

Figure 3.1: The estimated number of clusters (upper left) and estimated modularity (upper right) in LFR benchmark graphs with $N = 500$ nodes, an average degree of $d_{av} = 12$, comprising $c = 5$ clusters. The graphs are generated using parameters $\gamma = 2$ and $\beta_{lfr} = 3$ and varying the parameter μ. The lower left and lower right figures display the Normalized Mutual Information (NMI) and Element-centric similarity (ECS) measures for each clustering algorithm.

- A smaller power-law exponent indicates a heavy-tailed distribution. This means that there are more highly connected nodes (or "hubs") in the network. These networks are sometimes described as "scale-free" because their degree distributions are the same at all scales. In other words, the proportion of nodes with a certain number of links remains constant, regardless of the total number of links;
- A larger power-law exponent indicates a lighter-tailed distribution. This means that the network has fewer highly connected nodes. As the power-law exponent increases, the distribution becomes more similar to an exponential distribution, which has fewer hubs and is less skewed.

Therefore, based on this characteristic, we consider that as γ increases, the value of the α should also increase, since its network properties will be closer to a random network rather than a power-law network. We use LFR benchmark to generate networks with different γ to find the relationship between γ and α, by changing α and testing on networks with fixed γ to find the γ value that can make the algorithm perform optimally, and finally we summarize our findings as the following function:

$$\alpha = \begin{cases} 0.3 \times \gamma - 0.1 & \text{if } 1 < \gamma \leq 3.5 \\ 0.95 & \text{otherwise,} \end{cases} \tag{3.20}$$

Here we set an upper bound $\alpha = 0.95$ and the relationship between γ and γ for $1 < \gamma \leq 3.5$ is derived from a linear fit to the results of the above experiments. According to Figure 3.1, with the strategy of dynamically adjusting the value of γ, the performance of LCP$_n$ improves significantly from that with fixed α, but is clearly still inferior to that of non-backtracking.

It is worth pointing out that our strategy is not some optimal solution but only an empirical one. We still do not have a clear explanation for why the performance of LCP$_n$ differs so much from that of non-backtracking in power-law networks, even after adopting the above strategy. In addition, since γ

is a simple linear fit to the degree distribution of the power-law network, it is difficult to reflect the full characteristics of the corresponding network, which leads to the fact that the strategy of inferring α from γ is unlikely to be the optimal solution. Therefore, the reasons for the declining performance of spectral methods including $\mathrm{LCP}_{\mathrm{n}}$, non-backtracking, and eigengap on power-law networks still need further investigation.

4

Considered Clustering Algorithms

4.1. Modularity-based Methods

The subsequent three algorithms are categorized as modularity-based methods, indicating their pursuit of the optimal community partition through modularity optimization. It is essential to underscore, however, that the Louvain and Leiden methods employ quality functions that may not strictly adhere to the concept of modularity. For instance, Traag ***et al.*** highlight the Constant Potts Model (CPM) [9] [34] as an alternative quality function that mitigates the resolution limitations to a certain extent when contrasted with modularity.

4.1.1. Newman Method

Newman introduced a clustering algorithm based on modularity optimization [6]. The algorithm begins by estimating the bisection of a graph, aiming to generate the highest modularity value m using Equation (2.2), which can be expressed as:

$$m = \frac{1}{4L} y^T \cdot M \cdot y, \tag{4.1}$$

where, the $N \times 1$ vector y represents the cluster membership of each node, with values of either 1 or -1. The $N \times N$ modularity matrix $M = A - \frac{1}{2L} \cdot d \cdot d^T$ can be decomposed into its eigenvalues and eigenvectors as:

$$M = \sum_{i=1}^{N} \zeta_i \cdot z_i \cdot z_i^T, \tag{4.2}$$

where, the $N \times 1$ eigenvector z_i corresponds to the i-th eigenvalue ζ_i. The vector $y = \sum_{j=1}^{N} (z_j^T \cdot y) \cdot z_j$ can be expressed as a linear combination of the eigenvectors $\{z_i\}_{1 \leq i \leq N}$, transforming Equation (4.1) into:

$$m = \frac{1}{4L} \sum_{i=1}^{N} \zeta_i \cdot (z_j^T \cdot y)^2. \tag{4.3}$$

To maximize the modularity m, Newman proposed setting $y_i = 1$ if $(z_1)_i > 0$, otherwise $y_i = -1$. This procedure is repeated in subsequent iterations by dividing the graph into two partitions based on spectral properties. However, considering only the block sub-matrix of M corresponding to the cluster g in the next iteration would ignore inter-community links. Instead, for the estimated cluster g, the modularity matrix M_g is updated using:

$$M_g = m_{ij} - \left(\sum_{k \in g} m_{ik} \right) \cdot \delta_{ij}, \tag{4.4}$$

Here, the Kronecker delta δ_{ij} is 1 if $i = j$, and 0 otherwise. The algorithm terminates when further improvement in modularity m is no longer possible.

4.1.2. Louvain Method

The Louvain method, proposed by Blondel *et al.* [7], is a powerful yet straightforward heuristic clustering algorithm. This method employs an iterative and unsupervised two-step procedure to optimize modularity, denoted as m. The algorithm starts by assigning each node in a directed graph G with an $N \times N$ weighted adjacency matrix $\tilde{A}$ to its own distinct community. This forms the initial partition of the network.

In the first stage, the algorithm evaluates the change in graph modularity m if node i were to be assigned to the community of its neighboring node $j \in \mathcal{N}_i$. The modularity gain Δm resulting from assigning node i to community h of adjacent node j is defined in [7] as follows:

$$\Delta m = \left(\frac{\sum_{\text{in}} + 2\sum_{l:C_{lj}=1} \tilde{A}_{il}}{2L} - \left(\frac{\sum_{\text{tot}} + d_i}{2L} \right)^2 \right) - \left(\frac{\sum_{\text{in}}}{2L} - \left(\frac{\sum_{\text{tot}}}{2L} \right)^2 - \left(\frac{d_i}{2L} \right)^2 \right), \tag{4.5}$$

where $\sum_{\text{in}}$ represents the sum of the weights of intra-community links in community h, and $\sum_{\text{tot}}$ denotes the sum of the weights of all links in G incident to any node in community h.

During each iteration, the algorithm assesses whether moving node i to a neighboring community would result in a higher modularity value. If such a move is found, the node is moved to the community that provides the maximum modularity gain. If all computed gains Δm are either negative or smaller than a predefined small positive threshold value, node i remains in its original community. The first stage concludes when modularity m can no longer be increased by re-assigning nodes to neighboring communities. After completing a pass over all nodes, the algorithm compresses the network by merging communities that are connected by a single link, effectively creating a new, consolidated node for each merged community.

In the second stage of each iteration, the weighted graph obtained from the first stage is transformed into a new weighted graph, where each node represents a community. The link weight between two nodes h and g is equal to the sum of weights of all links between communities h and g in the graph obtained from the first stage. Moreover, the weight of a self-loop of node g in the new graph equals the sum of weights of all intra-community links in cluster g of the graph from the previous stage. This new graph is then fed into the first stage for the next iteration. The algorithm terminates when modularity m can no longer be increased. The time complexity of the Louvain method is linear with the number of links $\mathcal{O}(L)$ for sparse graphs [7].

4.1.3. Leiden Method

Although the Louvain method is widely used in clustering algorithms, it has a drawback of identifying poorly connected or disconnected communities. This limitation was first identified by Traag *et al.*, who introduced the Leiden algorithm as an improvement to the Louvain method in their work [9]. The Leiden algorithm aims to estimate graph partitions while ensuring the creation of connected communities. The Leiden algorithm involves three iterative steps:

- Local moving of nodes: This step is an enhanced version of the first step in the Louvain algorithm, described in Equation (4.5). While the Louvain algorithm randomly visits each node until modularity can no longer be improved by moving a node to a different community, the Leiden algorithm only visits nodes whose adjacent nodes have been relocated. This is achieved by placing nodes in a queue and iteratively checking if the cluster membership of a node can be updated to enhance the quality function. When a node is moved to another community, its neighbors from other communities are placed in the queue;
- Refinement of the partition: In this step, each node is initially assigned its own community. Nodes are merged locally, meaning only within communities estimated in the previous stage. Two nodes within the same community are merged only if both nodes are well connected to the community from the previous stage. At the end of the refinement stage, partitions from the first stage are often split into multiple communities;
- Aggregation of the network: This step involves aggregating the network based on the refined partition obtained from the previous stage, similar to the second stage of the Louvain algorithm.

The Leiden algorithm achieves faster clustering compared to the Louvain algorithm while generally providing improved partitions [9].

4.2. Spectral Methods

The following two algorithms are known as spectral methods. Neither of these algorithms possesses the inherent capacity for community partition. Given the community partitioning ability demonstrated by all the other algorithms and the resulting coherence in subsequent simulations, we resort to employing k-means clustering for community partitioning. In particular, these algorithms produce the number of clusters within the network. Leveraging this number along with the eigenvectors of the representation matrix of the network as inputs for the k-means clustering process, the resulting output furnishes the community partition for each node.

4.2.1. Non-backtracking Method

The non-backtracking matrix B is based on the concept of non-backtracking walks on a network G, which are walks where one does not immediately "reverse" along a link just traversed. In particular, a non-backtracking matrix B starts by creating a list of directed links for an undirected network, which means replacing every link with two directed links going in opposite directions. Therefore, for an undirected network $G(\mathcal{N}, \mathcal{L})$, the corresponding directed network is constructed with $2L$ links. Then, the $2L \times 2L$ non-backtracking matrix B is defined by:

$$B_{(u\to v),(w\to z)} = \begin{cases} 1 & \text{if } v = w \text{ and } u \neq z \\ 0 & \text{otherwise,} \end{cases} \tag{4.6}$$

where $v, w, z \in \mathcal{N}$.

Krzakala *et al.* [10] proposed a spectral method for clustering networks based on non-backtracking matrix. The non-backtracking matrix B, being asymmetric, typically has complex eigenvalues. In the complex plane, most of these eigenvalues are located within a bulk whose center is the origin and whose radius is the square root of the largest real eigenvalue. Therefore, they estimated the number of clusters within G corresponds to the quantity of real eigenvalues that lie outside this bulk. The computational complexity of non-backtracking method is $O(L^3)$, mainly for eigenvalue decomposition. And this complexity has the opportunity to be reduced to $O(N^3)$ [11].

4.2.2. Modularity Eigengap

Besides modularity as a quality function, Newman [6] also defined a $N \times N$ real symmetric matrix M with elements

$$M_{ij} = A_{ij} - \frac{d_i d_j}{2L} \tag{4.7}$$

which called the modularity matrix. The estimation of the number of clusters c is determined by the maximum eigengap of the modularity matrix M, following the same process described for the adjacency matrix A. The eigenvalues of the modularity matrix M can firstly be sorted in descending order $\lambda_1(M) \geq \lambda_2(M) \geq \cdots \geq \lambda_N(M)$. Hence, it can be observed that the eigenvalues of the modularity matrix M and the adjacency matrix A are interlaced [11]:

$$\lambda_1(A) \geq \lambda_1(M) \geq \lambda_1(A) \geq \lambda_2(M) \geq \cdots \geq \lambda_1(A) \geq \lambda_N(M) \tag{4.8}$$

Finally, the number of clusters c could be estimated by

$$c = \arg\max_i(\lambda_{i-1}(M) - \lambda_i(M)), \qquad i = 2, \ldots, N \tag{4.9}$$

where $\lambda_{i-1}(M) - \lambda_i(M)$ denotes the difference of two consecutive eigenvalues of modularity matrix M and estimated number of clusters c is equal to the index number of smaller eigenvalue in the maximum eigengap.

4.3. Local Dominance

Shang *et al.* proposed the Local Dominance [12] algorithm, designed to reveal the hidden hierarchy in the network by utilizing local information. Local Dominance starts with calculating the degree of each node. For a given node, a link pointing to an adjacent node will be established if the following criteria are met: the degree of the adjacent node is greater than or equal to the original node, and the degree of the adjacent node is the largest among all neighbors. Loops are not permitted in this process,

meaning that if a link already exists between two nodes, a second link in the opposite direction cannot be established.

Upon traversing all nodes, if any nodes possess multiple outgoing links, one is randomly preserved. Consequently, the algorithm transforms the initial network into a collection of tree structures, with each tree's root node referred to as the local leader and the leaf nodes as followers. In the context of Local Dominance, each tree represents a community. The total number of communities corresponds to the quantity of local leaders, and a local leader, along with their followers, constitutes the members of a community.

The time complexity of Local Dominance algorithm is linear in the number of links $O(L)$ [12], which could be one of the fastest community detection algorithms.

5 Results

5.1. Computational complexity

Table 5.1 shows the computational complexity of considered clustering algorithms:

Clustering Algorithm	Computational Complexity
Louvain[1]	$\mathcal{O}(L)$
Leiden	$\mathcal{O}(L)$
Local Dominance	$\mathcal{O}(L)$
LCP	$\mathcal{O}(N \cdot L)$
LCP_c	$\mathcal{O}(N \cdot L)$
LCP_n	$\mathcal{O}(N^3)$
Newman	$\mathcal{O}(N^3)$
Non-backtracking	$\mathcal{O}(N^3)$
Eigengap	$\mathcal{O}(N^3)$

Table 5.1: Computational Complexity of considered clustering algorithms

The Louvain method is known for its efficient clustering performance, requiring minimal computational complexity. Studies have demonstrated that its time complexity is linear with the number of links L for sparse graphs. An improved version of Louvain, called the Leiden method, achieves comparable complexity when performing community partitioning. On the other hand, the Local Dominance approach utilizes local information to estimate communities in a graph, resulting also in computational complexity that scales linearly with the number of links. In contrast, our LCP demands greater computational effort for estimating partitions. The computational complexity of LCP, as described in [13], scales linearly with the product of the number of nodes N and the number of links L. However, LCP_c, a variant of our LCP where the number of communities c is provided as input, exhibits comparable computational complexity. Furthermore, our LCP_n variant employs eigenvectors of the corresponding governing matrix to estimate communities. This approach requires computational effort that scales with the cube of the network size, denoted as $\mathcal{O}(N^3)$. This computational complexity also applies to any spectral clustering method, which is necessary for performing spectral decomposition.

5.2. SBM Benchmark

5.2.1. SSBM network with small number of clusters

SSBM network with $c = 2$ clusters

Figure 5.1 illustrates the clustering performance of the LCP algorithm and other methods considered in this thesis. Regarding the estimation of the number of communities c (top-left figure), the non-backtracking (NBT) method and the equivalent version of LCP (i.e. LCP_n, see Section 3.3) demonstrate superior performance, converging to two clusters for values of $b_{in} - b_{out}$ above the detectability

[1] Time complexity in case of sparse graphs.

threshold. The eigengap method follows closely, achieving convergence to the exact number of clusters for higher values of $b_{in} - b_{out}$, while estimating more than $c = 2$ communities in the undetectable regime. In terms of precision in estimating the true number of clusters, LCP ranks fourth and converges to $c = 2$ only when the clusters are clearly distinct (i.e., when there is a large difference between b_{in} and b_{out}), because it optimizes modularity m, as explained in the following paragraph. Although the Newman method performs the worst when the clusters are indistinguishable, it outperforms Louvain, Leiden and Local Dominance methods once the communities start to emerge from the randomness in the SSBM network structure. Louvain and Leiden methods perform similarly on SSBM networks with $c = 2$ clusters and converge to the correct number of clusters only in cases when there are almost no inter-community links. These two algorithms' tendency to identify a large number of communities stems from their design, which initializes every node as a separate community. Finally, Local Dominance performs the worst overall, irrespective of the number of inter-community links. In this SSBM model, the outcomes of Local Dominance exhibit a more robust correlation with the scale of the network. To illustrate, when considering a network comprising 500 nodes as depicted in these figures, the findings would typically manifest as approximately 20 clusters. Scaling up to 1000 nodes, this cluster count would elevate to approximately 40, assuming that other intrinsic network parameters, such as the average node degree, remain unchanged.

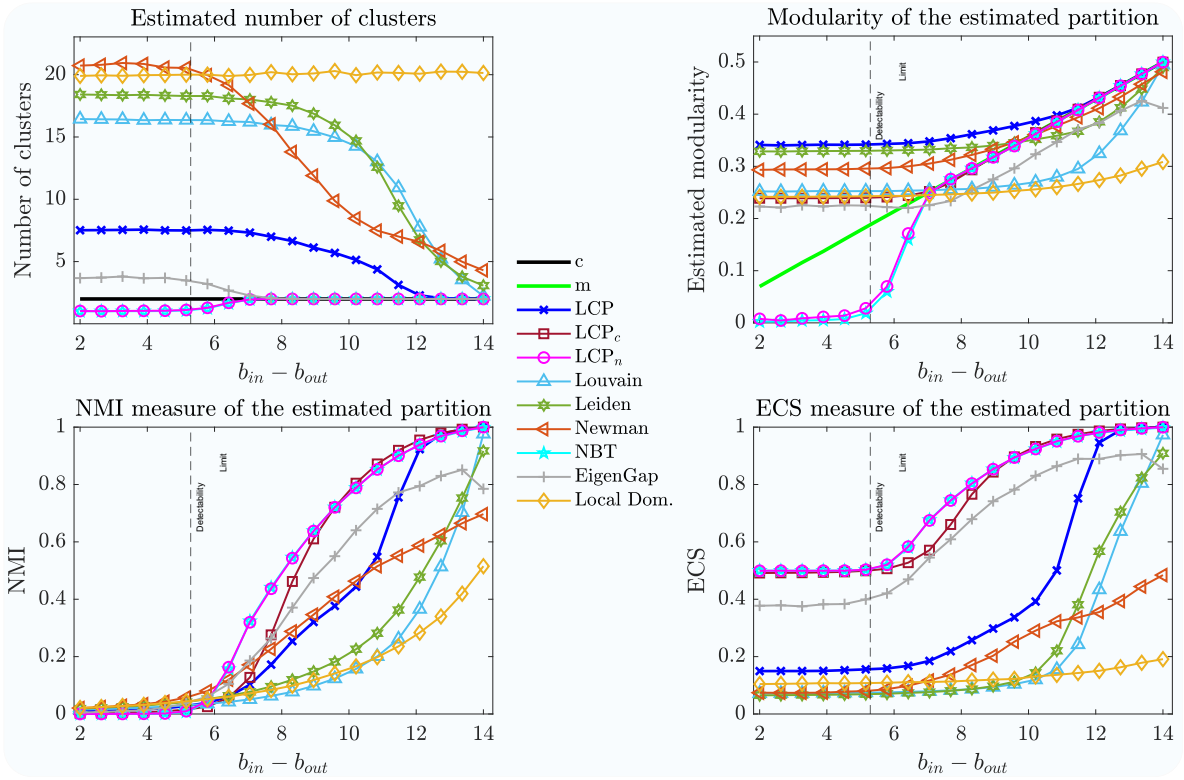

Figure 5.1: The estimated number of clusters (upper left-hand figure) and estimated modularity (upper right-hand figure) in SSBM graphs with $N = 500$ nodes, average degree $d_{av} = 7$ and $c = 2$ clusters, for different values of parameters b_{in} and b_{out}. The NMI and ECS measures per each clustering algorithm are provided in the lower left-hand side and lower right-hand side figures, respectively. The vertical dashed line indicates the clustering detectability threshold.

LCP identifies partitions with the highest modularity m overall, as depicted in the upper right-hand side of Figure 5.1. Even when original clusters are indistinguishable from randomness, LCP identifies alternative communities by optimizing modularity m. LCP$_c$, a variant of LCP which receives the true number of communities as input, achieves the original modularity m in the region where communities are visible. Furthermore, just above the detectability threshold, the modularity achieved by LCP$_c$ exceeds the original modularity m, indicating that alternative partitions with the same number of communities but higher modularity coexist around the threshold! The same argument explains the incorrect number of clusters c identified by LCP, as the achieved modularity is consistently higher than that of

the original partition. Non-backtracking method and our LCP_n achieve modularity m of the planted partition once they estimate the correct number of clusters c. In contrast, when original communities are indistinguishable from randomness, these methods fail to reveal any clusters, and thus the resulting modularity is zero. For lower values of $b_{in} - b_{out}$, the Leiden method performs closely to LCP. At the same time, the performance declines as the number of intra-community links increases compared to other algorithms. Leiden consistently outperforms Louvain, which aligns with the idea of the Leiden method as an improvement of the Louvain algorithm. As the Newman method converges to the correct number of clusters c, its modularity increases and tends towards the original modularity when clusters are visible while identifying alternative partitions with superior modularity when original clusters are not distinguishable. Finally, Louvain and Local Dominance perform similarly in modularity when original clusters are not clearly visible, while Louvain estimates better partitions as the number of inter-community links decreases.

We utilize the NMI measure explained in Section 2.1.2, together with the ECS measure described in Section 2.1.3, to compare the estimated partitions of each clustering algorithm considered with corresponding planted partitions. From the NMI measure presented in the bottom-left part of Figure 5.1, we observe that the non-backtracking method (NBT) and our LCP_n reveal partitions most similar to the planted partition overall, when they estimate a correct number of clusters. The LCP_c variant performs the best when clusters are visible (i.e. for higher values of $b_{in} - b_{out}$). However, just above the detectability threshold, LCP_c identifies alternative communities with superior modularity m and, thus, less similar to the planted partition. The Newman method and LCP perform similarly in the region above the detectability threshold but worse than the eigengap method. At the same time, LCP outperforms remaining considered algorithms, especially as the number of inter-community links decreases. Leiden and Louvain perform similarly, with Leiden reproducing original partitions with slightly higher accuracy, while Local Dominance performs the worst.

The element-centric similarity (ECS) measure, proposed in [16], resolves the bias in the NMI measure towards large number of clusters. The ECS measure reveals that NBT, our LCP_n and LCP_c, dominantly outperform other clustering algorithms when identifying partitions in the SSBM network with $c = 2$ communities, regardless of the number of inter-community links. The ECS measure achieved by LCP_c drops locally above the detectability threshold due to observed alternative partitions, as explained in the previous paragraph. Eigengap follows the aforementioned algorithms in ECS performance while being outperformed by our LCP in the case of a relatively low number of inter-community links. Overall, our LCP predominantly outperforms all non-spectral clustering methods considered. It is worth noting that LCP performs clustering based on a linear physical process while estimating the borders of clusters by maximizing modularity. Therefore, the ECS measure clearly indicates that LCP accurately discovers clusters through its underlying process. Other algorithms perform similarly, whereas Newman outperforms the remaining algorithms for the $b_{in} - b_{out}$ values above the detectability threshold. In contrast, the Leiden and Louvain algorithms dominate once the clusters become more visible.

Upon examining the distinct subfigures in isolation, a comprehensive analysis warrants their collective consideration. Evidently, a strong connection is discernible among the four subfigures. The behaviour of the algorithms in converging to the real values of the above four metrics is markedly different. For number of clusters, algorithms exhibit a progressive alignment with the real number of clusters as the $b_{in} - b_{out}$ expands, subsequently rendering the clusters more perceptible. In the context of modularity, this alignment materializes as a gradual trajectory toward and eventual alignment with the original modularity curve. Conversely, concerning NMI and ECS, this alignment is characterized by a pronounced leap in proximity to the convergence juncture, i.e., a realm where NMI and ECS values remarkably enhance within a compact span of the $b_{in} - b_{out}$.

While disparities persist in the behavioral tendencies of individual algorithms concerning their convergence towards the real values of the aforementioned four metrics, a noteworthy observation arises: the points of convergence remain uniform. Equally intriguing is the consistency displayed by algorithms based on similar ideas and techniques in terms of their convergent points. For instance, all three spectral methods converge slightly beyond the detectability threshold, with LCP exhibiting a delayed convergence, while the modularity-based algorithms approach convergence exclusively in scenarios where inter-community links are nearly absent. It's crucial to emphasize that across other simulations with different number of clusters, the coherence among these four subfigures remains unwavering. This consistency persists even in the face of potential variations in algorithmic performance and the timing of their individual convergences.

Furthermore, upon comparing ECS curves with NMI in the lower segment of the figure, a distinct pattern emerges. In the lower right quadrant of the figure, it becomes evident that nearly all algorithms have an initial value, i.e., their ECS values surpass 0 within the realm situated beneath the detectability threshold. Conversely, the NMI values for the corresponding range tend to converge towards a negligible 0. This phenomenon finds its roots in the inherent nature of ECS. For instance, consider the curve generated by the non-backtracking approach. Since the number of clusters estimated by non-backtracking is 1 in the region below the detectability threshold, i.e., the algorithm treats the whole network as a single cluster. Therefore the value of ECS will be $1/c$, where c represents the real value of the clusters in the network, in this case $c = 2$. Consequently, both non-backtracking and LCP$_n$ algorithms initiate their ECS curve with a value of 0.5. Notably, this stands in stark contrast to NMI, which, in the same scenario, attains a value of 0.

SSBM network with $c = 4$ clusters

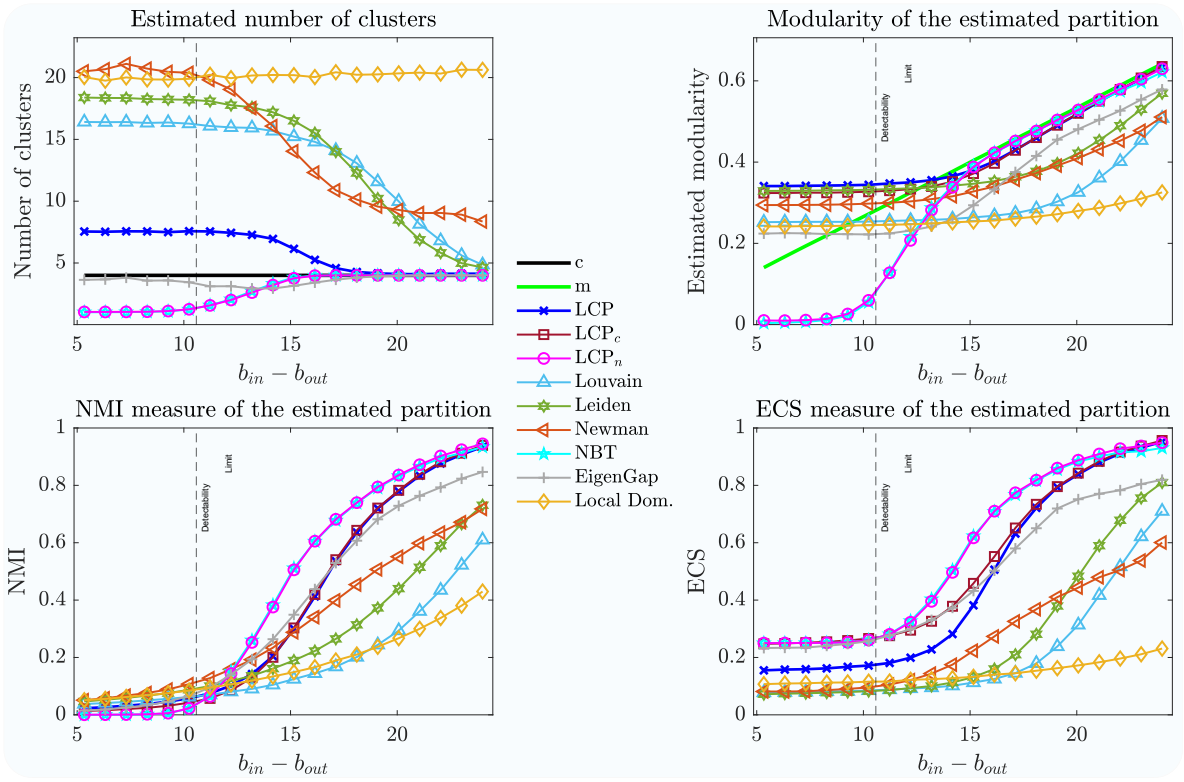

Figure 5.2: The estimated number of clusters (upper left-hand figure) and estimated modularity (upper right-hand figure) in SSBM graphs with $N = 500$ nodes, average degree $d_{av} = 7$ and $c = 4$ clusters, for different values of parameters b_{in} and b_{out}. The NMI and ECS measures per each clustering algorithm are provided in the lower left-hand side and lower right-hand side figures, respectively. The vertical dashed line indicates the clustering detectability threshold.

Figure 5.2 illustrates the clustering performance of our LCP and other considered clustering algorithms for the SSBM network with $c = 4$ communities. The rankings of clustering algorithms in accurately estimating the number of clusters remain consistent with those in the case of SSBM graphs having $c = 2$ communities. We observe that each considered clustering algorithm, in the case of SSBM networks with different $b_{in} - b_{out}$ values below the detectability threshold, tends to reveal the same number of clusters, regardless of the number of planted communities c. The algorithms, namely eigengap, non-backtracking method, and our LCP$_n$, exhibit the highest precision in estimating the number of communities, followed by LCP. Newman, Louvain, and Leiden methods demonstrate comparable performances, eventually converging to the correct value of c, particularly when the number of inter-community links is relatively low. However, Local dominance erroneously estimates the community structure irrespective of the number of links connecting nodes from different communities.

Our LCP produces partitions with the highest modularity overall, as depicted in the right-upper part of Figure 5.2. In this case, knowing the exact number of communities doesn't benefit LCP from higher modularity, as can be seen from the modularity performance of LCPc. Just above the detectability threshold, our LCP$_c$ reveals partitions different from the planted one, with the exact number of clusters but of superior modularity. NBT and our LCP$_n$ achieve modularity of the planted partition as their estimated number of clusters converges to the correct number of communities, outperforming eigengap consistently when communities are distinguishable from randomness. Leiden and Newman methods exhibit similar performances, followed by Louvain, while the Local Dominance algorithm performs the worst.

The lower part of Figure 5.2 illustrates the similarity between the estimated community structure of each clustering algorithm and the planted partition in the generated SSBM graph, using the NMI (left lower part) and ECS (right lower part) measures. Based on the NMI and ECS measures, NBT and our LCP$_n$ exhibit the highest similarity to the planted community structure overall for $b_{in} - b_{out}$ values above the detectability threshold. Their performance is followed by our LCP, which produces partitions significantly more similar to the original one than the remaining clustering algorithms considered. However, around and below the detectability threshold, the NMI measure of other clustering algorithms surpasses that of our LCP. This trend can be attributed to the bias of the NMI measure towards a larger number of clusters. Louvain, Leiden, Newman, and the Local dominance method estimate more clusters than our LCP, leading to higher NMI measures for lower values of $b_{in} - b_{out}$. As explained in Section 2.1.3, the ECS measure effectively mitigates the inherent bias towards a greater number of communities in the NMI measure. Consequently, the lower-right section of Figure 5.2 consistently demonstrates the superiority of our LCP across the entire range of $b_{in} - b_{out}$ values. Moreover, ECS provides us with valuable insights into the enhancement within LCP when guided by a predetermined number of clusters. The ECS curve corresponding to LCP$_c$ consistently maintains a higher position than that of the standard LCP until the latter approaches convergence with the actual clustering value. As the LCP gradually converges, the two curves subsequently align in a gradual manner.

5.2.2. SSBM network with large number of clusters

In contrast to the SSBM networks with $c = 2$ and $c = 4$ clusters, the modularity of the estimated partition by each considered algorithm for the range of $b_{in} - b_{out}$ values above the detectability limit are lower than the modularity m of the planted partition for $c = 8$, as depicted in the upper right part of Figure 5.3. This trend indicates that, as the number of clusters c increases, the planted partition possesses the highest modularity among all possible partitions on a given graph.

Regarding SSBM networks featuring 8 clusters, the performance of LCP exhibits a distinct character. In the top-left quadrant of Figure 5.3, it is evident that LCP closely approximates the true cluster value or converges to it consistently throughout. Conversely, in the remaining three quadrants, LCP's performance does not exhibit any remarkable tendencies. Generally, we expect algorithms to deliver consistent performance, implying similarity across various metrics. Specifically, when a clustering algorithm produces a number of clusters that approximates or precisely matches the actual value, we also anticipate its resultant community partition to exhibit high quality, thereby demonstrating proficiency across two metrics: NMI and ECS. A good example is that this phenomenon holds significance for spectral methods, as they solely furnish the cluster value, leaving the task of community partitioning to k-means clustering.

For LCP in this special scenario, by integrating this insight with other four simulations, we can deduce that the "default" value of output clusters for LCP is 8. In other words, LCP appears to lean towards presupposing the presence of 8 communities within the network, particularly when the delineation of communities is less distinct (around and below the detectability threshold). Besides, in the case of SSBM networks containing 8 or 10 clusters, the algorithms' performance shows a coherent pattern similar to that observed in simulations featuring networks with a smaller number of clusters. Nevertheless, it is noteworthy to highlight that the Louvain and Leiden algorithms achieve convergence to the true value notably sooner than in scenarios featuring a smaller number of clusters. When coupled with subsequent assessments involving even larger cluster numbers, these two modularity-based algorithms again proved to excel in managing scenarios characterized by cluster numbers that are neither excessively small nor overly large.

When the number of clusters grows to 20, most algorithms lose their clue about the true clustering according to Figure 5.5. As can be seen in the upper-left figure, only the three algorithms of the spectral

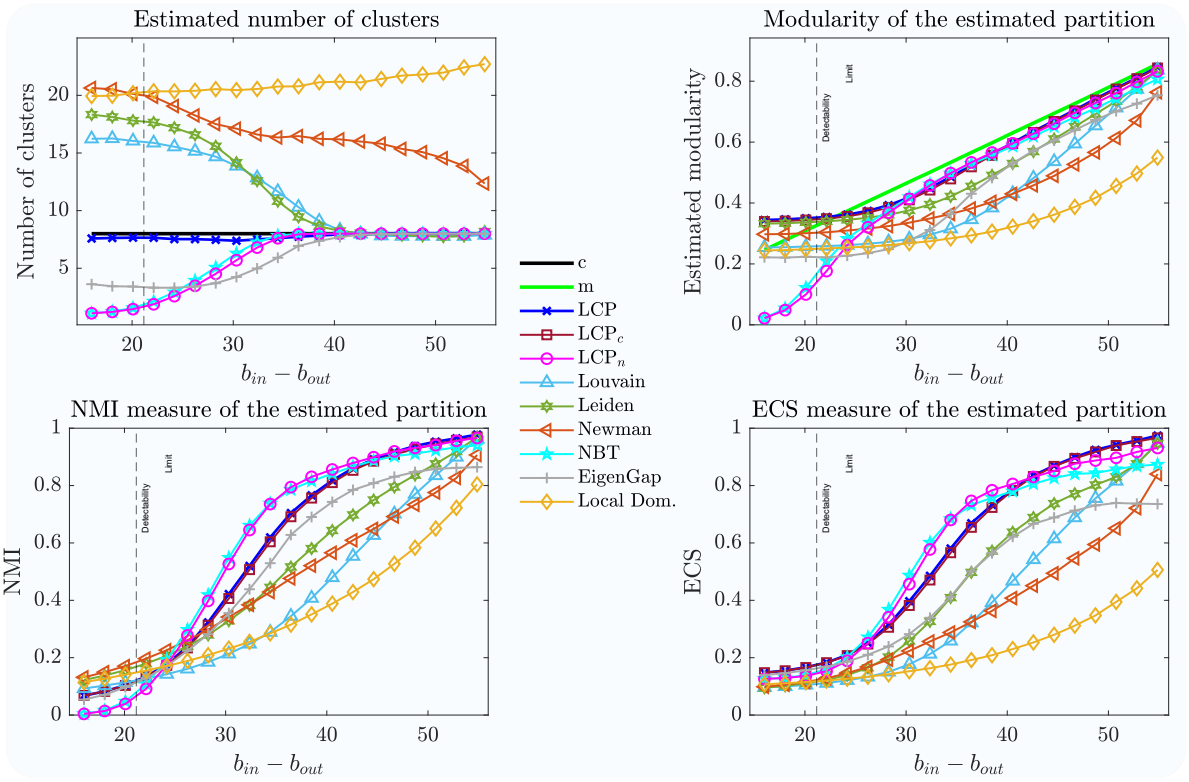

Figure 5.3: The estimated number of clusters (upper left-hand figure) and estimated modularity (upper right-hand figure) in SSBM graphs with $N = 500$ nodes, average degree $d_{av} = 7$ and $c = 8$ clusters, for different values of parameters b_{in} and b_{out}. The NMI and ECS measures per each clustering algorithm are provided in the lower left-hand side and lower right-hand side figures, respectively. The vertical dashed line indicates the clustering detectability threshold.

methods are able to converge to the true value after the clusters are already very well defined in the network. Other algorithms, including LCP, fail to converge to real value throughout this simulation, even when the difference $b_{in} - b_{out}$ is already so large that there are almost no inter-community links in the network. For modularity-based methods like Louvain and Leiden algorithms and the algorithm involving modularity such as LCP, poor performance is to be expected when the number of clusters becomes very large.

As the number of clusters is $c = 20$, the majority of algorithms appear to falter in discerning the accurate communities, as depicted in Figure 5.5. Notably, as evident in the upper-left chart, only the three algorithms based on the spectral method exhibit the capability to converge towards the true value, even after the network's clusters are well-defined. Conversely, other algorithms, including LCP, do not display an immediate propensity for converging to the true value. For methods grounded in modularity, such as the Louvain and Leiden algorithms, as well as the modularity-involved LCP, diminished performance becomes foreseeable when confronted with a substantially large number of clusters.

Modularity encounters a resolution limit [34], a phenomenon wherein the pursuit of optimizing modularity within extensive networks often proves inadequate in identifying small communities, even when their definition is unequivocal. This limitation arises from the fact that clusters characterized by minimal inter-community connections but exhibiting the utmost density of internal links-signifying the most discernible communities-may merge during modularity optimization, particularly in the context of sufficiently expansive networks [34]. This theory potentially elucidates the collective decline in performance among algorithms employing modularity in this particular scenario. Nonetheless, it is imperative to acknowledge that nearly all algorithms demonstrate varying degrees of performance degradation. Within the realm of community detection algorithms, the decrease in performance stemming from an abundance of clusters within a network could indeed signify an intrinsic limitation.

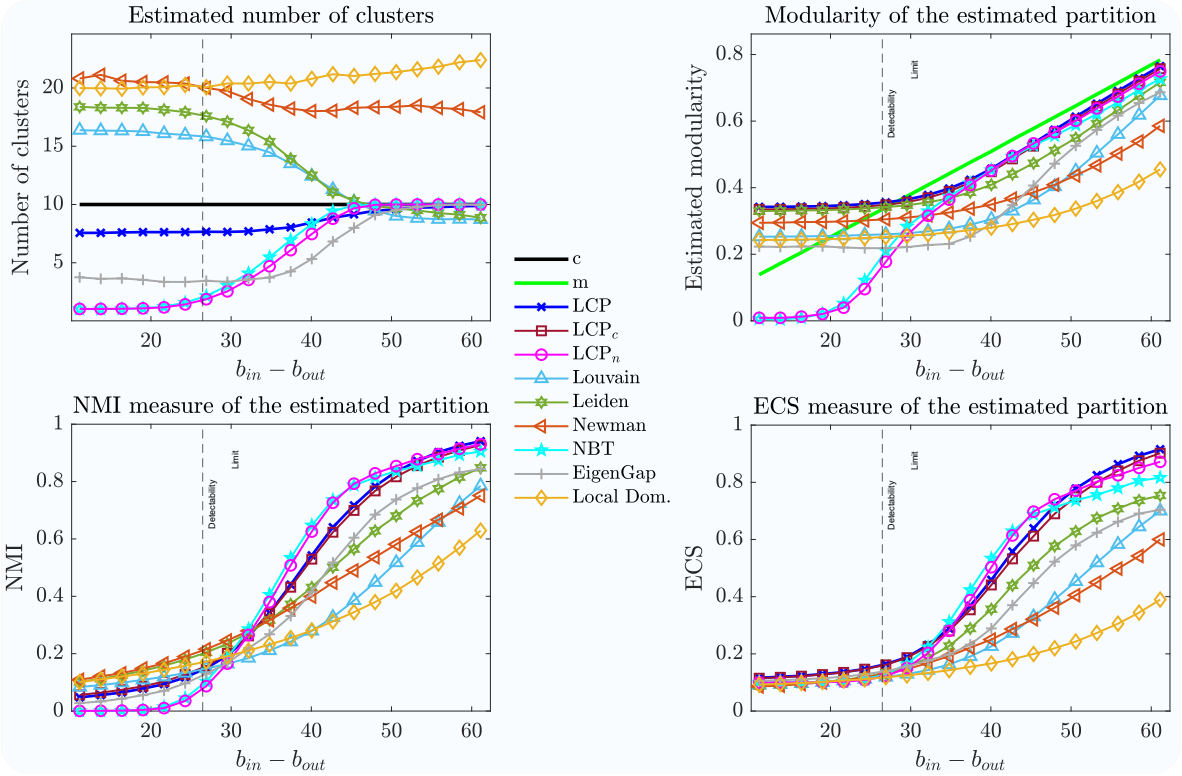

Figure 5.4: The estimated number of clusters (upper left-hand figure) and estimated modularity (upper right-hand figure) in SSBM graphs with $N = 500$ nodes, average degree $d_{av} = 7$ and $c = 10$ clusters, for different values of parameters b_{in} and b_{out}. The NMI and ECS measures per each clustering algorithm are provided in the lower left-hand side and lower right-hand side figures, respectively. The vertical dashed line indicates the clustering detectability threshold.

5.3. LFR Benchmark

The Lancichinetti-Fortunato-Radicchi (LFR) benchmark, proposed in [15], aims to overcome limitations in the SBM benchmark, including the uniform degree and community size distribution. The LFR benchmark incorporates power-law degree distribution and community size distribution inspired by observations in real-world networks. In this section, we evaluate the performance of our LCP method and other clustering algorithms under consideration.

Figure 5.6 illustrates the clustering performance of our LCP and other clustering algorithms considered in a network with $N = 500$ nodes and $c = 5$ communities. These communities were generated using the LFR model with parameters $\beta_{lfr} = 3$ and $\gamma = 2$. In the top left section of the figure, we observe that, overall, our LCP$_n$ outperforms all other methods in estimating the number of clusters. When the number of inter-community links is high, the NBT method incorrectly estimates the number of communities. Nevertheless, once the clusters become clearly visible in the network, NBT provides the most precise estimate among all the methods considered. Our LCP converges to the correct value of c for relatively low μ values, while identifying alternative community clusters elsewhere, which have a higher number of communities. In contrast, the eigengap method exhibits the opposite trend. It successfully reveals the true number of communities when there are relatively few connections between nodes from different clusters. However, as the ratio of inter-community links increases, it converges to a single community. The Louvain and Leiden methods perform similarly, with Leiden consistently providing a more accurate estimate of the true number of communities. Interestingly, both approaches tend to estimate an increasing number of communities as μ increases, until a certain value is reached. After this point, there is a decreasing trend in estimating c, indicating the presence of alternative community structures for large values of μ. We will further discuss this trend when analysing the achieved modularity m. Newman's method estimates an increasing number of communities as μ increases and reaches a saturation point around $c = 17$ for a large ratio of inter-community links. Finally, the Local

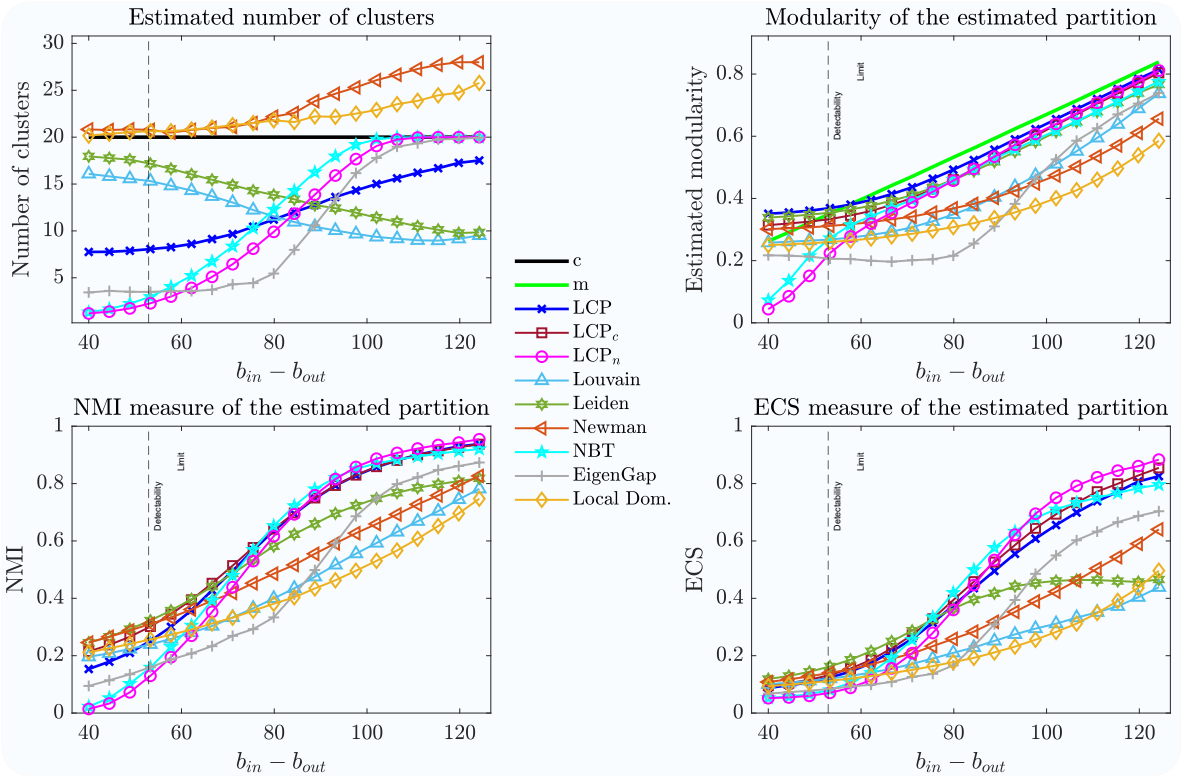

Figure 5.5: The estimated number of clusters (upper left-hand figure) and estimated modularity (upper right-hand figure) in SSBM graphs with $N = 500$ nodes, average degree $d_{av} = 7$ and $c = 20$ clusters, for different values of parameters b_{in} and b_{out}. The NMI and ECS measures per each clustering algorithm are provided in the lower left-hand side and lower right-hand side figures, respectively. The vertical dashed line indicates the clustering detectability threshold.

Dominance method fails to reveal any meaningful community structure across all the considered values of μ.

From the top-right portion of Figure 5.6, it is evident that our LCP provides the highest overall modularity of the estimated partition. The initial decreasing trend in modularity m indicates the increasing difficulty of accurately estimating the community structure as the ratio of inter-community links increases. However, for large values of μ, both our LCP and nearly all other clustering algorithms considered manage to recover community structures with higher modularity that that of the planted community structure. This trend clearly demonstrates the emergence of alternative community structures, distinct from the initially planted one, as μ values increase. When the number of communities is known, our LCP$_c$ performs slightly worse than LCP in terms of modularity but still surpasses the performance of other considered algorithms. This trend confirms the superiority of our LCP, where the proposed linear process of relocating nodes on a one-dimensional line successfully reveals communities with excellent accuracy. Leiden, NBT, Louvain, and Newman follow with comparable modularity performance, while our LCP$_n$ and eigengap exhibit significantly poorer results. Lastly, Local Dominance exhibits very low modularity m, converging to 0, which aligns with its failure to unveil any meaningful community structure for the majority of μ values.

From both the recorded NMI (bottom-left) and ECS (bottom-right) similarity measures presented in Figure 5.6, consistent patterns are observed for all the clustering algorithms considered. Whenever our LCP accurately estimates the true number of clusters, it exhibits the highest accuracy in recovering the original partition. However, for higher μ values, LCP starts revealing alternative communities, resulting in relatively lower values in both similarity measures. On the other hand, NBT demonstrates the slowest decrease in similarity measures as the ratio of inter-community links increases. Local Dominance performs the worst, while the remaining approaches achieve comparable results in terms of NMI and ECS values.

In case of LFR networks with $c = 10$ communities, the NBT variant of Linear Clustering Process

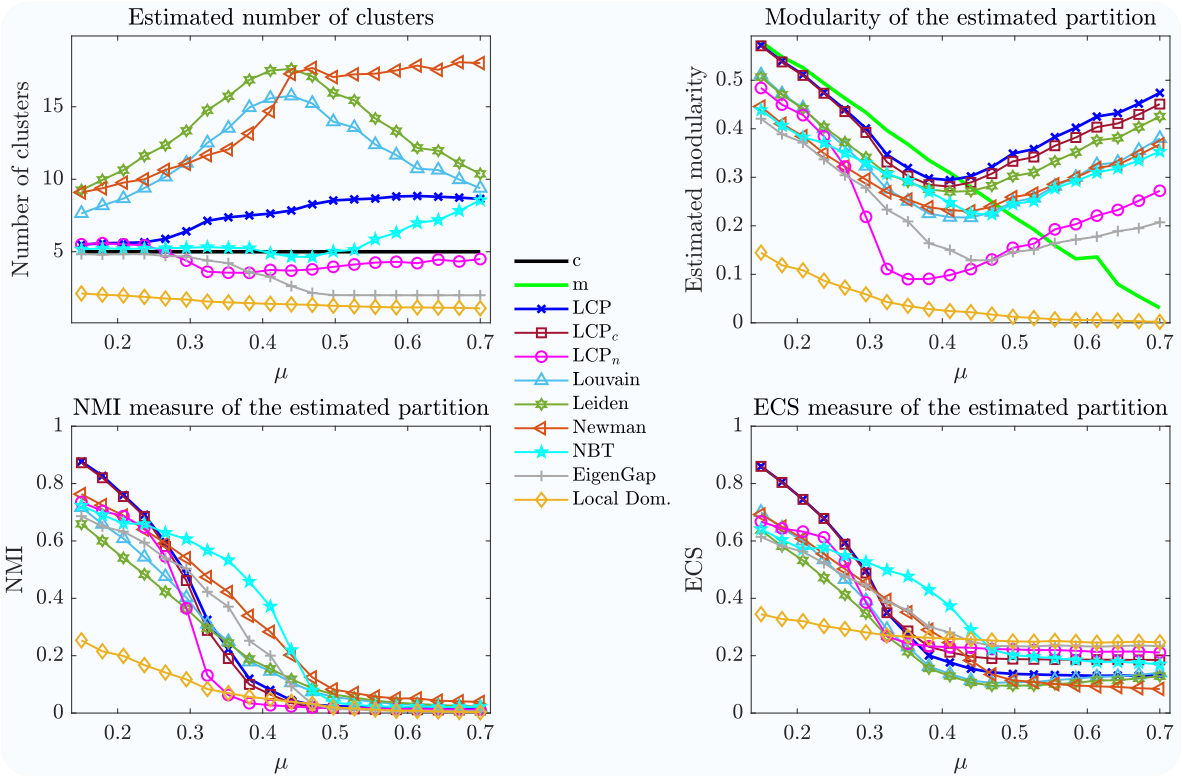

Figure 5.6: The estimated number of clusters (upper left) and estimated modularity (upper right) in LFR benchmark graphs with $N = 500$ nodes, an average degree of $d_{av} = 12$, comprising $c = 5$ clusters. The graphs are generated using parameters $\gamma = 2$ and $\beta_{lfr} = 3$ and varying the parameter μ. The lower left and lower right figures display the Normalized Mutual Information (NMI) and Element-centric similarity (ECS) measures for each clustering algorithm.

(LCP$_n$) performs significantly worse in estimating the number of communities compared to other spectral methods, namely NBT and eigengap. This is illustrated in the upper-left section of Figure 5.7. While LCP generally provides the most precise estimation of the number of clusters c, it does not necessarily generate community partitions that closely resemble the original planted partition in this particular case. This observation is supported by corresponding NMI and ECS similarity measures. As discussed in Section 2.2.1, for Stochastic Block Model (SSBM) networks, when LCP fails to identify the original partition, it tends to discover 8 communities, even though the true number of clusters is different. This behaviour persists and becomes more pronounced for larger values of the parameter μ, where LCP identifies communities that differ from the planted partition but eventually converges to 8 communities, regardless of the actual number of clusters c. Louvain outperforms both Leiden and Newman method consistently, while Local Dominance estimates communities with worst precision, and failing to discover any community for larger values of μ.

Our Linear Clustering Process (LCP) and LCP$_c$ yield partitions with the highest modularity overall, as shown in the upper-right portion of Figure 5.7. The performance of the other clustering methods considered is comparable to those obtained for LFR networks with $c = 5$ communities. The lower left (right) part of Figure 5.7 illustrates the similarity measures NMI (ECS) and demonstrates the dominance of our LCP within the range of μ values where it accurately estimates the number of communities. However, for μ values greater than 0.25, the NBT method outperforms our LCP in identifying the correct number of clusters and, consequently, produces partitions more similar to the planted community structures.

When the cluster count reaches $c = 20$, as shown in Figure 5.8, all employed algorithms exhibit a failure to converge towards the real number of clusters. This outcome further solidifies our analysis in Section 5.2.2, highlighting an inherent constraint wherein algorithmic efficacy deteriorates amidst a substantial proliferation of clusters within the network. This phenomenon transcends algorithmic categories; notable examples are modularity optimization algorithms, recognized for their resolution limitations. Meanwhile, we also note that when $c = 20$, the eigengap has the most significant perfor-

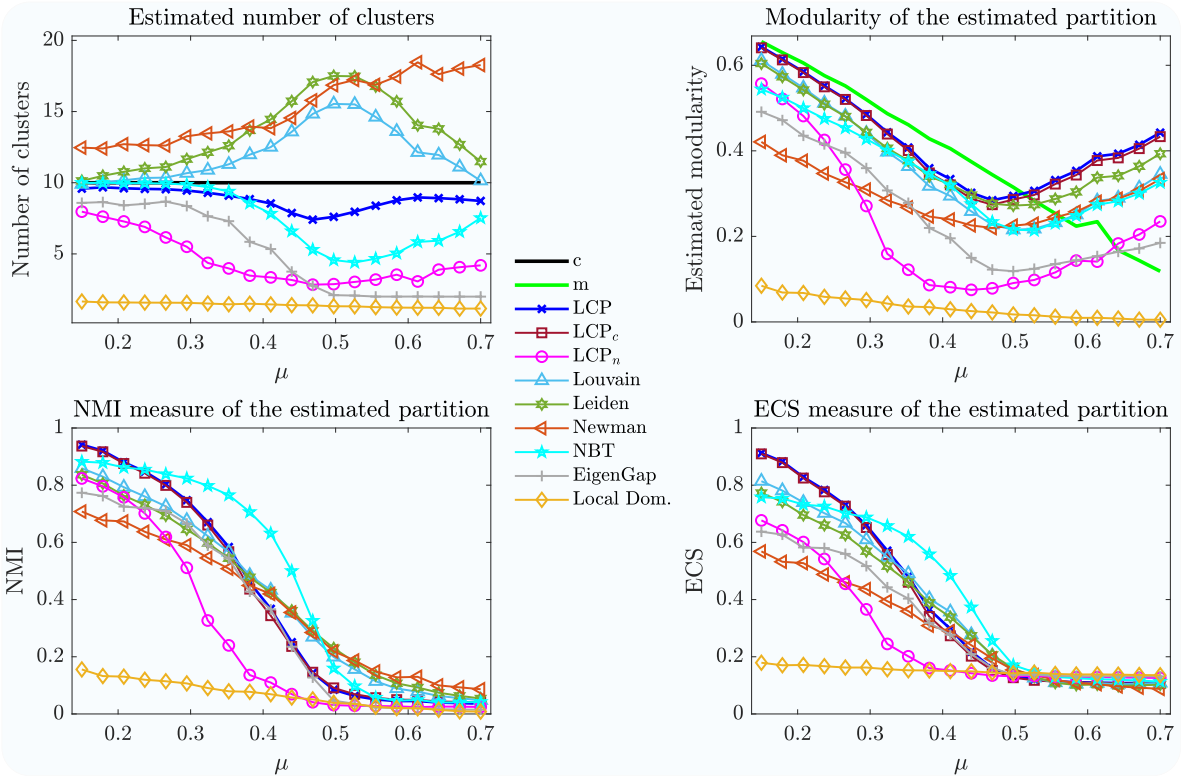

Figure 5.7: The estimated number of clusters (upper left) and estimated modularity (upper right) in LFR benchmark graphs with $N = 500$ nodes, an average degree of $d_{av} = 12$, comprising $c = 10$ clusters. The graphs are generated using parameters $\gamma = 2$ and $\beta_{lfr} = 3$ and varying the parameter μ. The lower left and lower right figures display the Normalized Mutual Information (NMI) and Element-centric similarity (ECS) measures for each clustering algorithm.

mance degradation. It performed slightly better than LCP$_n$ and worse than non-backtracking clustering in prior scenarios. However, in this simulation, it significantly lags behind LCP$_n$, underscoring that the spectral method is also susceptible to escalating cluster numbers.

In addition to the simulations on the LFR benchmark shown above, we also performed another set of simulations with $\gamma = 2.5$ and $\beta_{lfr} = 2.5$ while keeping other parameters consistent. Since the performance of all considered algorithms on this set of simulations is indistinguishable from the aforementioned, we refrain from further elaboration and present the results in Figure 5.9, 5.10, and 5.11.

5.4. Random network models

Erdős-Rényi random graphs [26] generally lack a clear and well-defined community structure, as demonstrated in [35, p. 630] by the average clustering coefficient, a measure of the ratio of the number of links between a node's neighbours and the maximum possible number of links. Due to the random nature of connections between nodes, the links tend to be evenly distributed across the nodes without observable community structure. However, community detection algorithms can still be applied to uncover potential structures or patterns within these graphs.

The upper-left part of Figure 5.12 illustrates the estimated number of clusters c by LCP and other considered clustering algorithms on ER graphs with $N = 500$ nodes and varying link density p_{ER}. Apart from the non-backtracking (NBT) method and our LCP$_n$, which accurately recover the exact number of communities, the remaining algorithms provide incorrect estimates. LCP and eigengap exhibit similar performance, while Local Dominance and Newman methods perform comparatively poorer in estimating the number of communities but with increased precision as the link density p_{ER} rises. Notably, Louvain and Leiden algorithms estimate a higher number of communities, which increases with the link density p_{ER}.

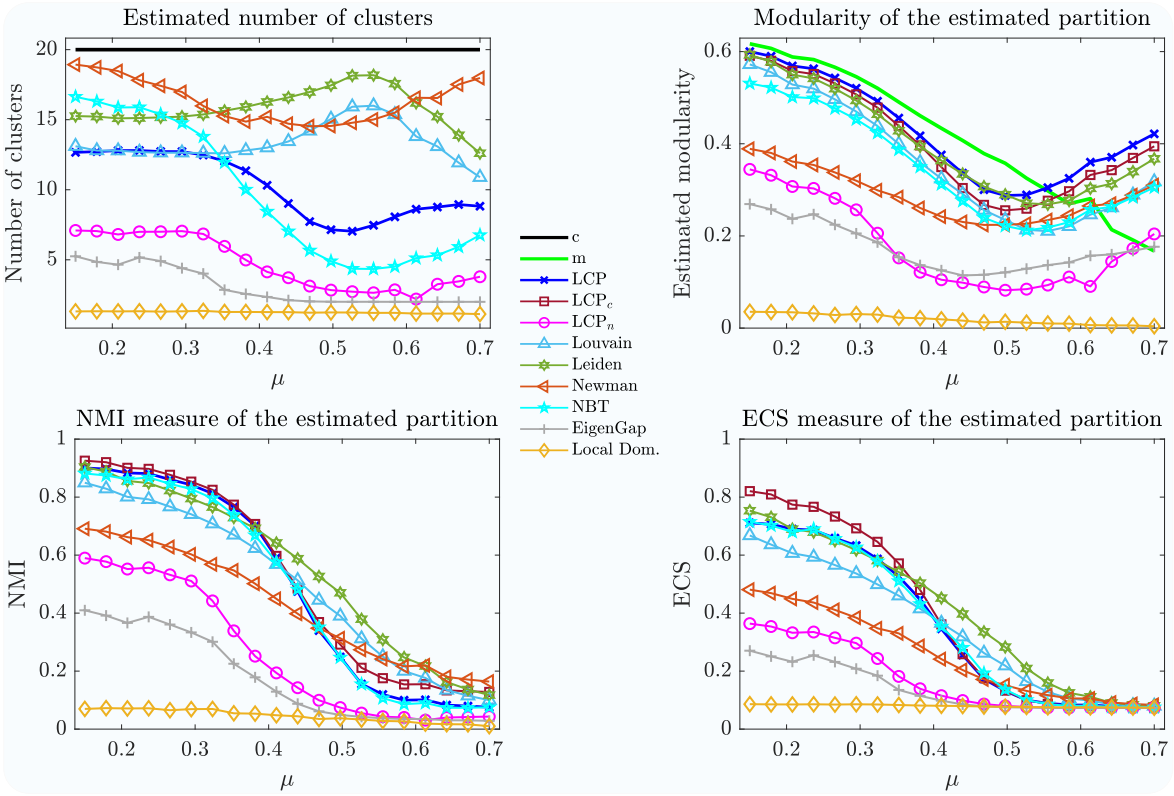

Figure 5.8: The estimated number of clusters (upper left) and estimated modularity (upper right) in LFR benchmark graphs with $N = 500$ nodes, an average degree of $d_{av} = 12$, comprising $c = 20$ clusters. The graphs are generated using parameters $\gamma = 2$ and $\beta_{lfr} = 3$ and varying the parameter μ. The lower left and lower right figures display the Normalized Mutual Information (NMI) and Element-centric similarity (ECS) measures for each clustering algorithm.

Regarding the achieved modularity m of the estimated partition, our LCP algorithm surpasses other analysed clustering algorithms for all considered values of link density p_{ER}. Newman and Leiden algorithms rank as the second and third best in terms of modularity performance, while Local Dominance performs the worst. Overall, clustering algorithms reveal a community structure with higher modularity in sparse ER graphs compared to graphs with higher link density. Although ER graphs, on average, do not exhibit a community structure, specific instances of the graph may contain more visible clusters. The probability of finding communities in ER graphs is greater when the graph is sparser, as an increased number of links makes the graph more regular in nature, causing communities to dissipate.

Interestingly, when applied to the Barabási-Albert graphs, the Local Dominance algorithm provides cluster estimates with a precision closely resembling that of the non-backtracking method (NBT) and our LCP$_n$, as shown in the upper middle section of Figure 5.12. On the other hand, our LCP algorithm estimates a significantly lower number of clusters than the other three algorithms, gradually decreasing the number of discovered clusters as the parameter m_{BA} increases. The Newman method ranks fifth in the estimated number of communities c, also exhibiting a decrease in the number of communities as the parameter m_{BA} increases. In contrast, the Louvain and Leiden algorithms, while optimizing modularity m, estimate an increasing number of clusters c that does not result in the highest modularity m, as evidenced by the LCP performance in the lower middle section of Figure 5.12. We observe that Local Dominance and our LCP$_n$ achieve the lowest modularity m, followed by NBT and eigengap. As with ER graphs, our LCP algorithm achieves the highest modularity overall. Newman, Louvain, and Leiden methods perform similarly in modularity m, which decreases as the parameter m_{BA} of the BA model increases.

The Watts-Strogatz graph model, proposed in [29], is derived from a regular ring lattice by randomly rewiring connections while maintaining a fixed number of nodes N and links L. By increasing the rewiring probability p_{WS}, a small-world phenomenon emerges abruptly, as illustrated by the average

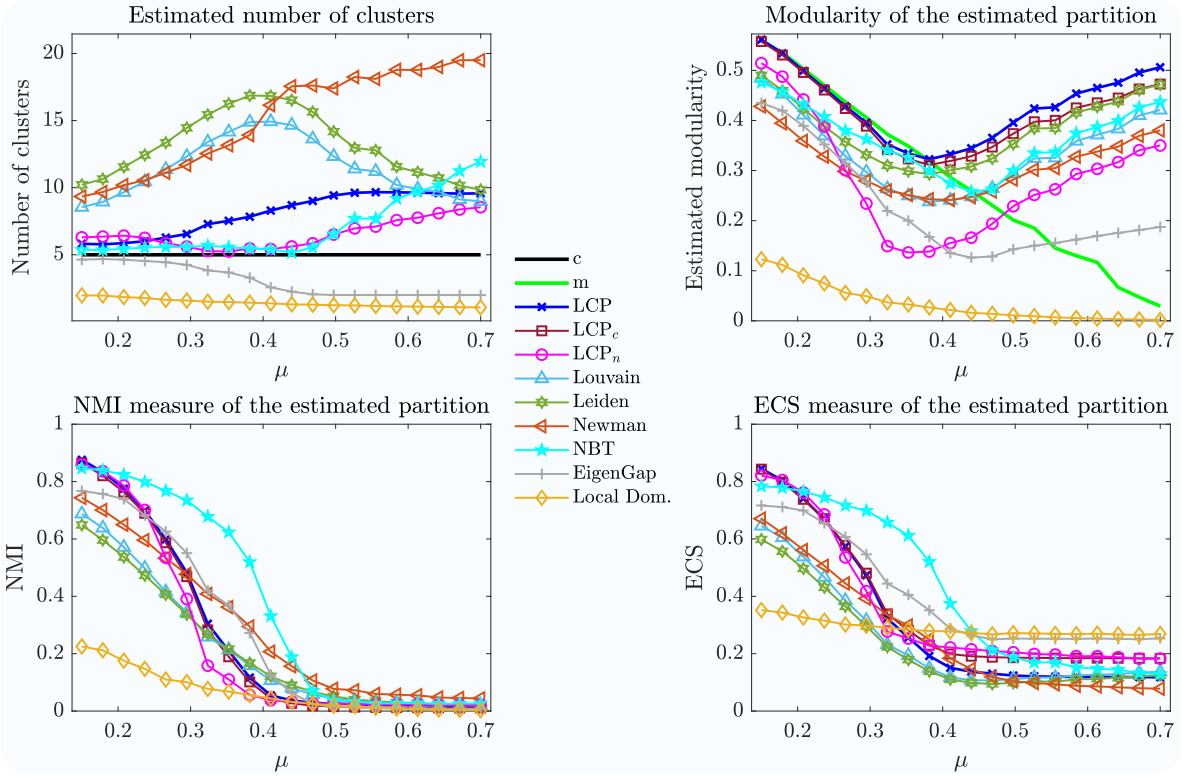

Figure 5.9: The estimated number of clusters (upper left) and estimated modularity (upper right) in LFR benchmark graphs with $N = 500$ nodes, an average degree of $d_{av} = 12$, comprising $c = 5$ clusters. The graphs are generated using parameters $\gamma = 2.5$ and $\beta_{lfr} = 2.5$ and varying the parameter μ. The lower left and lower right figures display the Normalized Mutual Information (NMI) and Element-centric similarity (ECS) measures for each clustering algorithm.

shortest path hop count on a logarithmic scale in [29, Figure 2]. However, this phenomenon is not observed locally, as demonstrated by the clustering coefficient (defined in [35, page 630]). The right-hand part of Figure 5.12 the number of clusters c (upper figure) and achieved modularity m (lower figure) of considered clustering algorithms, when applied to the Watts-Strogatz model with $N = 500$ nodes, using different rewiring probability p_{WS}.

For low values of p_{WS}, the Local Dominance method fails to detect any community structure, resulting in a low modularity value. As the rewiring probability p_{WS} increases, the Local Dominance method gradually reveals an increasing number of communities with modularity levels comparable to other algorithms. Surprisingly, the eigengap method underperforms by estimating the highest number of communities c, thus achieving the lowest modularity m values. Overall, the Louvain and Leiden methods estimate the lowest number of communities, leading to lower modularity m values than the remaining methods. Our LCP records fewer communities than the remaining clustering methods while achieving the highest modularity overall. Finally, we highlight that our LCP$_n$ consistently outperforms the non-backtracking (NBT) method.

5.5. Real-world networks

5.5.1. Real-world networks without ground-truth communities

Table 5.2 provides a summary of the clustering performance of our LCP algorithm and other clustering algorithms from existing literature on real-world networks where the ground-truth community partition is unknown. The network sizes range from $N = 34$ (Karate networks) to $N = 1589$ (Co-authorship network). Among the seven networks considered, our LCP algorithm achieves the highest modularity m in four cases, while it ranks as the second best in the Dolphins and Football networks. In one case, specifically the Co-authorship network, it ranks as the fourth best algorithm in terms of modularity.

Due to the absence of real community partitions, our analysis was confined to a basic assessment of

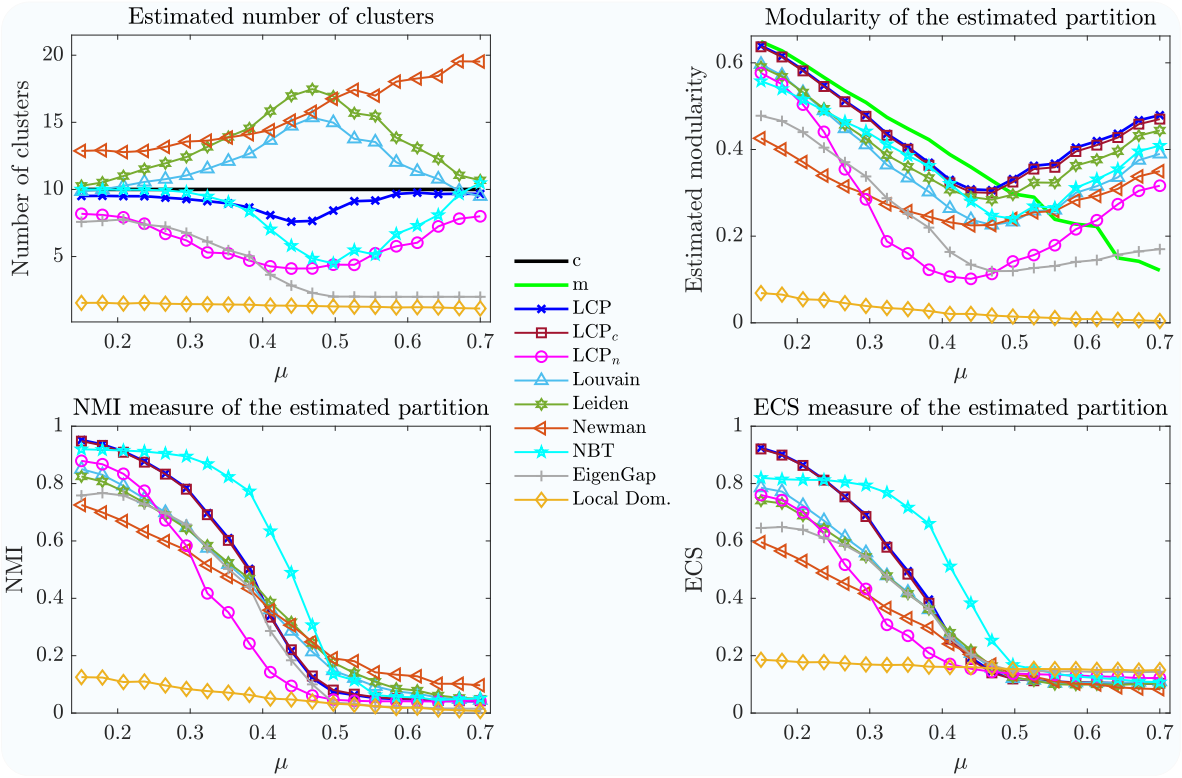

Figure 5.10: The estimated number of clusters (upper left) and estimated modularity (upper right) in LFR benchmark graphs with $N = 500$ nodes, an average degree of $d_{av} = 12$, comprising $c = 10$ clusters. The graphs are generated using parameters $\gamma = 2.5$ and $\beta_{lfr} = 2.5$ and varying the parameter μ. The lower left and lower right figures display the Normalized Mutual Information (NMI) and Element-centric similarity (ECS) measures for each clustering algorithm.

algorithmic performance centered around modularity. In the next section we consider several networks featuring ground-truth communities, mainly social network and citation networks, to do further analysis of the algorithm's performance on real-world networks.

5.5.2. Real-world networks with ground-truth communities

In this section, we evaluate the performance of our Linear Clustering Process (LCP) and other clustering algorithms on real-world networks with known ground-truth community structures. The Email-EU network comprises $N = 1005$ nodes and has the highest link density, with a total of $L = 25571$ links. This network forms $c = 42$ communities. Our LCP identifies a partition with the highest modularity, while the Leiden algorithm estimates more accurately the number of clusters, resulting in a partition that closely aligns with the ground truth. The Cora network consists of $N = 2708$ nodes interconnected by $L = 5429$ links, forming $c = 7$ communities. The Leiden algorithm produces a community structure with the highest modularity, closely followed by our LCP. However, our LCP excels in estimating the number of communities compared to Leiden. Finally, the Citeseer network exhibits the smallest number of clusters ($c = 6$). While the Newman algorithm identifies a partition with the highest modularity, its estimated number of communities deviates significantly from the correct value.

Based on the results presented in Table 5.3, we note a significant sensitivity of the NMI and ECS similarity measures to the estimated number of clusters. The NMI measure exhibits a bias towards partitions with a larger number of communities. Conversely, for the Citeseer network, we observe that the ECS measure of both the Louvain and eigengap algorithms is similar, despite their substantially different estimates of the number of clusters. This corroborates that ECS is a fairer metric than NMI for clustering similarity.

Overall, finding the real number of clusters and ground-truth community partitions present a notable challenge for these algorithms since the original modularity of all three networks is relatively low. While a majority of algorithms, particularly those based on modularity, manage to uncover alternative

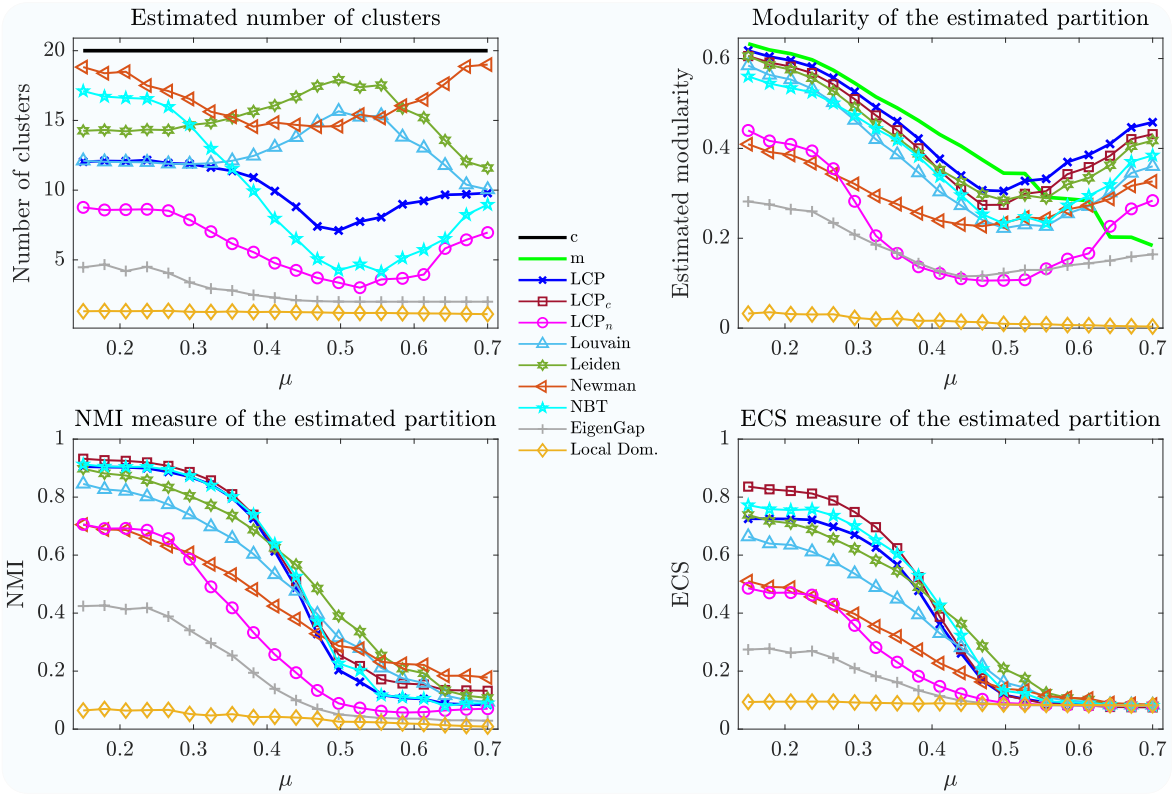

Figure 5.11: The estimated number of clusters (upper left) and estimated modularity (upper right) in LFR benchmark graphs with $N = 500$ nodes, an average degree of $d_{av} = 12$, comprising $c = 20$ clusters. The graphs are generated using parameters $\gamma = 2.5$ and $\beta_{lfr} = 2.5$ and varying the parameter μ. The lower left and lower right figures display the Normalized Mutual Information (NMI) and Element-centric similarity (ECS) measures for each clustering algorithm.

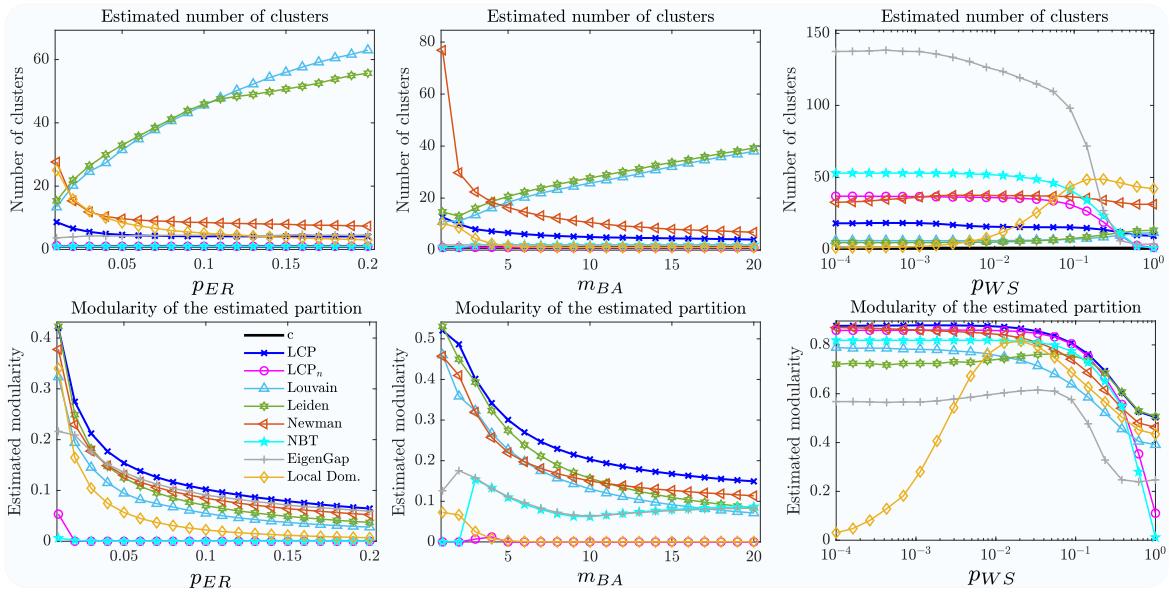

Figure 5.12: The estimated number of clusters (upper figures) and estimated modularity (bottom figures) in Erdős-Rényi random graph (left-hand side) with varying link density p_{ER}, Barabási-Albert graph (middle part) with varying parameter m_{BA}, and Watts-Strogatz graph (right-hand side) with varying rewiring probability p_{WS}. The graphs consist of $N = 500$ nodes.

Network	N	L	LCP		Louvain		Leiden		Newman	
			c	m	c	m	c	m	c	m
Karate	34	78	3	**0.3922**	4	0.3565	4	0.3729	5	0.3776
Dolphins	62	159	4	0.5057	4	0.4536	5	**0.5105**	6	0.4894
Polbooks	105	441	3	**0.5160**	4	0.4897	4	0.5026	8	0.4160
Football	115	613	7	0.5894	7	0.5442	7	0.5635	11	0.4623
Facebook	347	2519	8	**0.4089**	16	0.3726	18	0.3792	23	0.3770
Polblogs	1490	19090	19	**0.4224**	7	0.3385	11	0.3117	4	0.3459
Co-authorship	1589	2742	40	0.9296	272	0.9423	270	0.9410	28	0.7393
Network	N	L	Local Dom.		NBT		LCP_n		Eigengap	
			c	m	c	m	c	m	c	m
Karate	34	78	2	0.3123	2	0.3715	1	0.0000	2	0.2780
Dolphins	62	159	3	0.3620	2	0.3698	2	0.3698	2	0.3115
Polbooks	105	441	2	0.4451	3	0.5085	2	0.4546	2	0.4167
Football	115	613	6	0.3205	10	**0.5939**	5	0.5522	11	0.5927
Facebook	347	2519	8	0.2067	8	0.3638	7	0.3544	2	0.2836
Polblogs	1490	19090	3	0.2799	8	0.2149	5	0.3480	2	0.2679
Co-authorship	1589	2742	277	**0.9431**	23	0.5005	17	0.5806	2	0.1288

Table 5.2: Clustering performance of our LCP and considered existing clustering algorithms on real-world networks without ground-truth communities

partitions boasting superior modularity compared to the original partition, this deficiency is perceptibly accentuated in the case of LCP. This heightened emphasis on LCP's limitations is accentuated by previous simulations that underscored its superior proficiency in optimizing modularity relative to other algorithms.

Network	N	L	C	M	LCP				Louvain			
					c	m	nmi	ecs	c	m	nmi	ecs
Email-EU	1005	25571	42	0.2880	9	**0.3860**	0.5466	0.2513	12	0.3795	0.5530	0.2747
Cora	2708	5429	7	0.6401	25	0.7296	0.3138	0.2121	86	0.6775	0.3691	**0.2808**
Citeseer	3264	9072	6	0.5042	65	0.8027	0.1399	0.0737	394	0.7722	0.3095	0.1878
Network	N	L	C	M	Leiden				Newman			
					c	m	nmi	ecs	c	m	nmi	ecs
Email-EU	1005	25571	42	0.2880	17	0.3745	**0.6352**	**0.3711**	14	0.3492	0.5668	0.3003
Cora	2708	5429	7	0.6401	85	**0.7403**	0.4201	0.2722	68	0.7166	0.4146	0.1896
Citeseer	3264	9072	6	0.5042	395	0.7367	0.3438	**0.2064**	311	**0.8276**	0.3307	0.0560
Network	N	L	C	M	Local Dom.				NBT			
					c	m	nmi	ecs	c	m	nmi	ecs
Email-EU	1005	25571	42	0.2880	1	0.0000	0.0000	0.0669	17	0.2792	0.5061	0.2599
Cora	2708	5429	7	0.6401	181	0.6728	**0.4271**	0.1648	40	0.5000	0.3016	0.1558
Citeseer	3264	9072	6	0.5042	505	0.7691	**0.3889**	0.0619	17	0.3595	0.1618	0.1594
Network	N	L	C	M	LCP_n				Eigengap			
					c	m	nmi	ecs	c	m	nmi	ecs
Email-EU	1005	25571	42	0.2880	2	0.0932	0.2310	0.1211	5	0.2931	0.3538	0.1673
Cora	2708	5429	7	0.6401	39	0.5674	0.3320	0.1943	2	0.1072	0.1599	0.1961
Citeseer	3264	9072	6	0.5042	16	0.4389	0.1573	0.1581	3	0.1583	0.0960	0.1760

Table 5.3: Clustering performance of our LCP and considered existing clustering algorithms on real-world networks with ground-truth communities

6

Conclusions and Future Work

6.1. Conclusions

In this thesis, we conducted a comprehensive analysis of the clustering efficacy exhibited by the recently introduced Linear Clustering Process (LCP) on various benchmarks and networks. Our thesis entailed a comparative assessment of LCP against several well-established clustering algorithms, each employing distinct techniques for partition estimation, including modularity optimization, spectral methods and hierarchical method. Our simulation results consistently reveal that, across a range of scenarios involving both synthetic benchmarks and real-world networks, LCP consistently yields high-quality community partitions while maintaining computational complexity comparable to the simplest existing clustering algorithms.

Among the considered clustering algorithms, the LCP algorithm consistently demonstrates superior modularity. When the network exhibits clearly discernible clusters, LCP excels at precisely recovering partitions that closely mirror the actual clusters. In other words, LCP identifies partitions with high NMI and ECS measures. Impressively, even in scenarios where clusters are obscured by random patterns, LCP adeptly identifies alternative clusters that showcase enhanced modularity, albeit with lower NMI and ECS values.

Moreover, the ECS metric sheds light on how the real number of clusters, when utilized as input, enhances the quality of LCP's partitions. Notably, the ECS value of LCP_c surpasses that of the original LCP when the latter hasn't yet converged to the actual cluster count. In a variety of cases, LCP_n achieves results on par with the non-backtracking method, with the exception of power-law networks. This achievement is coupled with a clear and coherent underlying process and rationale. Furthermore, our proposed strategy substantially narrows the performance gap between LCP_n and the non-backtracking method on power-law networks, although a slight disparity persists. This presents a significant advancement over the original LCP_n approach, highlighting the efficacy of our strategy.

Finally, it's worth noting that we've uncovered intriguing insights beyond just the LCP algorithm. The Local Dominance algorithm emerges as a standout performer only on a few specific real-world networks, yet it falters when faced with other network types and synthetic benchmarks. Meanwhile, spectral methods demonstrate remarkable consistency, particularly in yielding top-tier community partitions when able to precisely estimate cluster numbers. Capitalizing on this attribute, they dominate networks like SSBM and those sharing similar characteristics. There is a slight decline in their effectiveness observed on power-law networks, although the overall performance remains impressive.

Altogether, our findings underscore the remarkable effectiveness of LCP in fostering highly modular and meaningful community structures across diverse network settings.

6.2. Future Work

Although this thesis addresses some problems, there are still issues that need to be addressed in future work for both algorithms and simulations.

This thesis offers a possibility to comprehend non-backtracking clustering. As showed earlier, the LCP_n approach demonstrates comparable performance to the non-backtracking method across the majority of networks. This observation offers a suggestion that the non-backtracking matrix may not

inherently possess exceptional characteristics. Our grasp of the underlying principles behind non-backtracking clustering remains limited, primarily based on observational insights. Thus, the identification of a novel matrix exhibiting analogous behaviors could prove instrumental in elucidating these principles. Moreover, the discernible performance contrast between LCP_n and non-backtracking techniques on power-law networks serves as a promising avenue for deeper insight into the workings of the non-backtracking method.

As for simulations, exhaustive testing has been conducted, scrutinizing algorithmic performance across networks encompassing a wide array of distinct features. Notwithstanding these endeavors, there linger unexplored terrains primed for future investigation. Among these, the exploration of networks that deviate from equilibrium, such as imbalanced Stochastic Block Model (SBM) networks, holds significant promise. Within this thesis, the focus has predominantly rested on Symmetric SBM networks, characterized by equitably sized communities. However, the forthcoming inquiry will delve into the algorithm's capacity to discern minute clusters within the framework of asymmetrical configurations, where community sizes vary. A reasonable anticipation is that the efficacy of algorithms grounded in modularity principles may encounter challenges within such networks compared with SSBM networks, attributable to the inherent limitations of modularity when grappling with intricate resolutions. Notably, the LFR benchmark networks considered in this paper also exhibit an inherent imbalance. However, due to the distinct nature of power-law networks generated by the LFR benchmark as compared to the SBM network, a direct comparison proves elusive.

Besides, in the realm of network clustering and community detection, traditional machine learning methods have found extensive application. However, the landscape is evolving with an escalating fascination for the seamless incorporation of deep learning techniques. Among these, the emergence of deep neural networks, notably graph neural networks (GNNs), has sparked significant interest due to their remarkable proficiency in deciphering intricate relationships and latent patterns within networks. The potential gains from this integration are profound, with the tantalizing prospect of elevating community detection and clustering accuracy to unprecedented levels.

The pursuit of innovation in this field extends to the exploration of hybrid models, constituting a fusion of time-tested traditional machine learning algorithms and cutting-edge deep learning methodologies. This dual-pronged approach holds immense potential, capitalizing on the individual merits of each paradigm. By amalgamating the interpretability and robustness of traditional methods with the feature extraction prowess of deep learning, these hybrid models stand poised to unlock enhanced performance and finer-grained community detection. This confluence is not merely an endeavor to harness the strengths of diverse techniques; it signifies a strategic endeavor to push the boundaries of network analysis, fostering a symbiotic relationship between conventional wisdom and modern intelligence. As the research horizon expands, these hybrid paradigms could potentially transcend existing limitations, fueling a new era of accuracy and insight in network clustering and community detection.

References

[1] Albert-László Barabási. “Network science”. In: *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences* 371.1987 (2013), p. 20120375.

[2] Mark Newman. *Networks*. Oxford university press, 2018.

[3] Santo Fortunato. “Community detection in graphs”. In: *Physics reports* 486.3-5 (2010), pp. 75–174.

[4] Satu Elisa Schaeffer. “Graph clustering”. In: *Computer Science Review* 1.1 (2007), pp. 27–64. ISSN: 1574-0137. DOI: https://doi.org/10.1016/j.cosrev.2007.05.001. URL: https://www.sciencedirect.com/science/article/pii/S1574013707000020.

[5] Mark EJ Newman and Michelle Girvan. “Finding and evaluating community structure in networks”. In: *Physical review E* 69.2 (2004), p. 026113.

[6] Mark EJ Newman. “Modularity and community structure in networks”. In: *Proceedings of the national academy of sciences* 103.23 (2006), pp. 8577–8582.

[7] Vincent D Blondel et al. “Fast unfolding of communities in large networks”. In: *Journal of statistical mechanics: theory and experiment* 2008.10 (2008), P10008.

[8] Andrea Lancichinetti and Santo Fortunato. “Community detection algorithms: A comparative analysis”. In: *Phys. Rev. E* 80 (5 Nov. 2009), p. 056117. DOI: 10.1103/PhysRevE.80.056117. URL: https://link.aps.org/doi/10.1103/PhysRevE.80.056117.

[9] Vincent A Traag, Ludo Waltman, and Nees Jan Van Eck. “From Louvain to Leiden: guaranteeing well-connected communities”. In: *Scientific reports* 9.1 (2019), p. 5233.

[10] Florent Krzakala et al. “Spectral redemption in clustering sparse networks”. In: *Proceedings of the National Academy of Sciences* 110.52 (2013), pp. 20935–20940. DOI: 10.1073/pnas.1312486110. eprint: https://www.pnas.org/doi/pdf/10.1073/pnas.1312486110. URL: https://www.pnas.org/doi/abs/10.1073/pnas.1312486110.

[11] Gabriel Budel and Piet Van Mieghem. “Detecting the number of clusters in a network”. In: *Journal of Complex Networks* 8.6 (Mar. 2021). cnaa047. ISSN: 2051-1329. DOI: 10.1093/comnet/cnaa047. eprint: https://academic.oup.com/comnet/article-pdf/8/6/cnaa047/36510011/cnaa047.pdf. URL: https://doi.org/10.1093/comnet/cnaa047.

[12] Fan Shang et al. “Local dominance unveils clusters in networks”. In: *ArXiv* abs/2209.15497 (2022).

[13] Ivan Jokić and Piet Van Mieghem. “Linear Clustering Process on Networks”. In: *IEEE Transactions on Network Science and Engineering* (2023), pp. 1–10. DOI: 10.1109/TNSE.2023.3271360.

[14] Paul W Holland, Kathryn Blackmond Laskey, and Samuel Leinhardt. “Stochastic blockmodels: First steps”. In: *Social networks* 5.2 (1983), pp. 109–137.

[15] Andrea Lancichinetti, Santo Fortunato, and Filippo Radicchi. “Benchmark graphs for testing community detection algorithms”. In: *Physical review E* 78.4 (2008), p. 046110.

[16] Alexander J Gates et al. “Element-centric clustering comparison unifies overlaps and hierarchy”. In: *Scientific reports* 9.1 (2019), p. 8574.

[17] Leon Danon et al. “Comparing community structure identification”. In: *Journal of statistical mechanics: Theory and experiment* 2005.09 (2005), P09008.

[18] Maximilian Jerdee, Alec Kirkley, and M. E. J. Newman. *Normalized mutual information is a biased measure for classification and community detection*. 2023. arXiv: 2307.01282 [cs.SI].

[19] Ulrik Brandes et al. “On modularity clustering”. In: *IEEE transactions on knowledge and data engineering* 20.2 (2007), pp. 172–188.

[20] Piet Van Mieghem et al. “Spectral graph analysis of modularity and assortativity”. In: *Physical Review E* 82.5 (2010), p. 056113.

[21] Roger A Horn and Charles R Johnson. *Matrix analysis*. Cambridge university press, 2012.

[22] Aurelien Decelle et al. “Inference and phase transitions in the detection of modules in sparse networks”. In: *Physical Review Letters* 107.6 (2011), p. 065701.

[23] Aurelien Decelle et al. “Asymptotic analysis of the stochastic block model for modular networks and its algorithmic applications”. In: *Physical Review E* 84.6 (2011), p. 066106.

[24] Lev Muchnik et al. “Origins of power-law degree distribution in the heterogeneity of human activity in social networks”. In: *Scientific reports* 3.1 (2013), p. 1783.

[25] Gergely Palla et al. “Uncovering the overlapping community structure of complex networks in nature and society”. In: *nature* 435.7043 (2005), pp. 814–818.

[26] Paul Erdős, Alfréd Rényi, et al. “On the evolution of random graphs”. In: *Publ. Math. Inst. Hung. Acad. Sci* 5.1 (1960), pp. 17–60.

[27] Piet van Mieghem. *Graph Spectra for Complex Networks*. Cambridge University Press, 2010. DOI: 10.1017/CBO9780511921681.

[28] Réka Albert and Albert-László Barabási. “Statistical mechanics of complex networks”. In: *Rev. Mod. Phys.* 74 (1 Jan. 2002), pp. 47–97. DOI: 10.1103/RevModPhys.74.47. URL: https://link.aps.org/doi/10.1103/RevModPhys.74.47.

[29] Duncan J Watts and Steven H Strogatz. “Collective dynamics of ‘small-world’networks”. In: *nature* 393.6684 (1998), pp. 440–442.

[30] Omer Angel, Joel Friedman, and Shlomo Hoory. “THE NON-BACKTRACKING SPECTRUM OF THE UNIVERSAL COVER OF A GRAPH”. In: *Transactions of the American Mathematical Society* 367.6 (2015), pp. 4287–4318.

[31] Aaron Clauset, Cosma Rohilla Shalizi, and M. E. J. Newman. “Power-Law Distributions in Empirical Data”. In: *SIAM Review* 51.4 (2009), pp. 661–703. DOI: 10.1137/070710111. eprint: https://doi.org/10.1137/070710111. URL: https://doi.org/10.1137/070710111.

[32] Aaron Clauset, Cosma Rohilla Shalizi, and Mark EJ Newman. “Power-law distributions in empirical data”. In: *SIAM review* 51.4 (2009), pp. 661–703.

[33] Albert-László Barabási and Réka Albert. “Emergence of Scaling in Random Networks”. In: *Science* 286.5439 (1999), pp. 509–512. DOI: 10.1126/science.286.5439.509. eprint: https://www.science.org/doi/pdf/10.1126/science.286.5439.509. URL: https://www.science.org/doi/abs/10.1126/science.286.5439.509.

[34] J. M. Kumpula et al. “Limited resolution in complex network community detection with Potts model approach”. In: *The European Physical Journal B* 56.1 (Mar. 2007), pp. 41–45. ISSN: 1434-6036. DOI: 10.1140/epjb/e2007-00088-4. URL: https://doi.org/10.1140/epjb/e2007-00088-4.

[35] Piet Van Mieghem. *Performance analysis of complex networks and systems*. Cambridge University Press, 2014.

A

Nomenclature

A.1. List of Abbreviations

Abbreviation	Definition
LCP	Linear Clustering Process
NMI	Normalized Mutual Information
SBM	Stochastic Block Model
SSBM	Symmetric Stochastic Block Model
LFR	Lancichinetti-Fortunato-Radicchi
ECS	Element-Centric Similarity
NBT	Non-backtracking
ER	Erdős-Rényi
BA	Barabási-Albert
WS	Watts-Strogatz
LCP_c	LCP for a known number of communities
LCP_n	Non-backtracking variant of LCP
CPM	Constant Potts Model
GNNs	Graph Neural Networks

A.2. List of Notations

Notation	Explanation
G	Graph
$\mathcal{N}$	Set of N nodes of graph G
$\mathcal{N}_i$	Set of neighboring nodes of node i
$\mathcal{L}$	Set of L links of graph G
N	Number of nodes in graph G
L	Number of links in graph G
A	Adjacency matrix of graph G
a	Element of adjacency matrix A
d	Degree vector
d_i	Degree of node i
Δ	Degree diagonal matrix
u	$N \times 1$ all-one vector
m	Modularity
C	Cluster matrix
c	Number of clusters

Notation	Explanation
n_i	Number of nodes in cluster i
F	Confusion matrix
$I_n(P_0, P_e)$	Normalized mutual information metric
P_0	Known partition
P_e	Estimated partition
c_0	Number of known clusters
c_e	Number of estimated clusters
p_{ij}	Personalized PageRank
δ	Kronecker delta function
c_γ, c_β	Cluster
v_i, v_j	Element of corresponding cluster
α_{ecs}	Restart probability constant
$\mathcal{A}$, $\mathcal{B}$	Clustering
$S(\mathcal{A}, \mathcal{B})$	Element-centric similarity score
p_{in}, b_{in}	Intra-community link probability
p_{out}, b_{out}	Inter-community link probability
$E[D]$	Expected degree
γ	Power-law exponent of degree distribution
β_{lfr}	Power-law exponent of community size distribution
μ	Inter-community link probability
d_{av}	Average degree
p_{ER}	Link density
m_0	Initial number of nodes
m_{BA}	Number of nodes connected by the new node
p_i	Probability that the new node is connected to node i
$P(d)$	Degree distribution
k_{WS}	Number of the nearest neighbors
p_{WS}	Rewiring probability
α	Attraction strength
δ	Repulsion strength
$x_i[k]$	Position of node i at time k
k	Discrete time
I	$N \times N$ identity matrix
W	$N \times N$ topology-based matrix
β	$N \times 1$ eigenvalue vector
β_i	Eigenvalue i
Y	$N \times N$ orthogonal eigenvector matrix
y_i	Eigenvector i
$\hat{y}_i$	Sorted eigenvector i
r	Ranking vector
R	Permutation matrix
$\hat{A}$	$N \times N$ relabeled adjacency matrix
$\hat{d}$	$N \times 1$ relabeled degree vector
S	$N \times N$ scaling matrix
υ	A small positive value
M_c	Aggregated modularity matrix
g, h	Estimated clusters
ν	$(2c - 1 \times 1)$ vector
ν_g	Element of vector ν
B	$2L \times 2L$ non-backtracking matrix
B^*	$2N \times 2N$ matrix
O	$N \times N$ all-zero matrix
$f_D(d)$	Probability density function of degree d
c_p	Normalization constant

Notation	Explanation
y	Cluster membership
M	$N \times N$ modularity matrix
z_i	i-th eigenvector
ζ_i	i-th eigenvalue
$\tilde{A}$	$N \times N$ weighted adjacency matrix
Δm	Modularity gain
$\sum_{\text{in}}$	Sum of the weights of intra-community links in community h
$\sum_{\text{tot}}$	Sum of the weights of all links in G incident to any node in community h
$\lambda(A)$	Eigenvalues of the adjacency matrix A
$\lambda(M)$	Eigenvalues of the modularity matrix M

B

Source Code

B.1. Linear Clustering Process

B.1.1. Original LCP

```
function [C,c,m] = LCP(A)
% This function implements the Linear Clustering Process (LCP), from the
% publication
N = size(A,1);                                                          % Number of nodes
N_pr = 30;                                                              % Number of iterations
L_per = 60/N_pr;                                                        % Ratio of links whose weight will be scaled
L_rem = linspace(L_per,L_per,N_pr);                                     % Scaling intensity for links in each iteration

alpha = 0.95;                                                           % Attraction strength
delta = 1e-3;                                                           % Repulsion strength

A_fm = A;                                                               % Store the adjacency matrix

Inds = zeros(N,N_pr);                                                   % Initalise the ranking of each node in y_2 per iteration
Pos_M = zeros(N,N_pr);                                                  % Initalise the position of each node in y_2 per iteration
m_it = zeros(N_pr,1);                                                   % Initalise the modularity in each iteration
C_it = zeros(N,N_pr);                                                   % Initalise the number of clusters in each iteration
c_it = zeros(N_pr,1);                                                   % Initalise the modularity in each iteration

W = compute_W(A_fm,N,alpha,delta);                                      % Compute W matrix (we compute it only once!)
counter = 1;
while(counter <= N_pr)                                                  % Perform N_pr iterations of LCP
    A_model = eye(N) + A_fm.*W - diag(A_fm.*W*ones(N,1));               % Compute state space matrix
    [ind_r,pos_r,y_2] = compute_y_2(A_model);                           % Compute the position vector in the steady state
    %     [ind_r,pos_r,y_2] = compute_y_2_v2(A_model);                  % Compute the position vector in the steady state
    Inds(:,counter) = ind_r; Pos_M(:,counter) = pos_r;                  % Store the position of each node per iteration
    [~,m_it(counter),C_it(:,counter),c_it(counter)] = optimize_m(A(ind_r,ind_r));       % Identify the clusters, for given position vector
```

```
28      %     [A_fm,~,~] = scale_links(A_fm,ind_r,L_rem(counter),counter,N_pr,y_2);          %
            Scale the weights of the identified inter-community links links and recompute the
            adjacency matrix
29      C = ones(N,1)*C_it(:,counter)' - C_it(:,counter)*ones(1,N); C = (abs(C) > 0);
30      [A_fm,~,~] = scale_links_v2(A_fm,ind_r,L_rem(counter),counter,N_pr,y_2,C);          %
            Scale the weights of the identified inter-community links links and recompute the
            adjacency matrix
31      counter = counter + 1;                                                              %
            Update the counter
32  end
33  [val_mod,ind] = max(m_it);                                                              %
        Adopt the iteration with the highest modularity
34  c = c_it(ind);                                                                          %
        Output the estimated number of clusters
35  m = val_mod;                                                                            %
        Output the estimated modularity
36  C = zeros(N,1);
37  C(Inds(:,ind)) = C_it(:,ind);                                                           %
        Output the cluster membership function of each node
38
39  return                                                                                  %
        Return C and m
40  end
41
42  %% Used sub-functions
43  function [A_new,A_rem,Deg_new] = scale_links(A,ind,trsh,br_it,N_pr,y_2)
44  % Adjacency matrix A provided as input is not relabeled!
45  N = size(A,1);                                                  % Network size
46  A_n = A(ind,ind);                                               % Relabeled adjacency
        matrix A based on y_2 (eq. 10, page 11)
47  Deg_v = A_n*ones(N,1);                                          % Relabeled degree vector
         based on y_2 (eq. 10, page 11)
48  N_rem_1 = ceil(nnz(A)/2*trsh/100);                              % Number of links, whose
        weight to scale wihtin one iteration
49  % Dist_M_1 = (abs(diag(1:N)*A_n - A_n*diag(1:N))).*A_n;          % Compute the distance
        matrix between any two adjacent nodes (sec 5.1, page 15)
50  Dist_M_1 = (abs(diag(1:N)*A_n - A_n*diag(1:N))).*A_n;           % Compute the distance
        matrix between any two adjacent nodes (sec 5.1, page 15)
51  A_new = A_n;                                                    % Store the adjacency
        matrix
52  counter_1 = 1;                                                  % Coutner for the number
        of links with scaled weight
53  while counter_1 < N_rem_1                                       % Run the loop until
        N_rem links are scaled in weight
54      tr = 1;                                                     % Initialise the
            indicator function in case the link is found.
55      while (tr)                                                  % Run the loop until the
            link is removed
56          val = max(Dist_M_1(:));                                 % Find the maximum
                distance value
57          [ind_1,ind_2] = find(Dist_M_1 == val);                  % Identify the link
                connecting two most far away nodes
58          if((Deg_v(ind_1(1))) > 1 && (Deg_v(ind_2(1))) > 1 ...
59                  && A_new(ind_1(1),ind_2(1)) == 1)                   % Check the validity
                        of a link (i.e. if adjacent nodes have degree > 1)?
60              Dist_M_1(ind_1(1),ind_2(1)) = 0;                    % Define the distance of
                    a chosen link as 0, so it is not conisdered anymore
61              Dist_M_1(ind_2(1),ind_1(1)) = 0;
62              counter_1 = counter_1 + 1; tr = 0;                  % Increment the counter
63              Deg_v(ind_1(1)) = Deg_v(ind_1(1)) - 1;              % Update the degree
                    values for each node
64              Deg_v(ind_2(1)) = Deg_v(ind_2(1)) - 1;
65              A_new(ind_1(1),ind_2(1)) = (0.01/(N_pr))*br_it;     % Scale the weight of a
                    link
66              A_new(ind_2(1),ind_1(1)) = (0.01/(N_pr))*br_it;
67              A_new(ind_1(1),ind_2(1)) = 0;       % Scale the weight of a link
68              A_new(ind_2(1),ind_1(1)) = 0;
69          else                                                    % This link can not be
                removed
70              Dist_M_1(ind_1(1),ind_2(1)) = 0;                    % Define the distance of
                    a chosen link as 0, so it is not conisdered anymore
```

```
            Dist_M_1(ind_2(1),ind_1(1)) = 0;
        end
    end
end
A_new(ind,ind) = A_new;
A_rem = (A - A_new);
Deg_new = A_new*ones(N,1);
% Check if there are zero-degree nodes (testing the code)
if(nnz(Deg_new) < N)
    disp('Number of zero degree nodes:')
    disp(length(Deg_new) - nnz(Deg_new))
    plot(sort(Deg_new))
    error('Something is wrong!!!')
end
end

function [A_new,A_rem,Deg_new] = scale_links_v2(A,ind,trsh,br_it,N_pr,y_2,C)
% Adjacency matrix A provided as input is not relabeled!
N = size(A,1);                                                  % Network size
A_n = A(ind,ind);                                               % Relabeled adjacency
    matrix A based on y_2 (eq. 10, page 11)
Deg_v = A_n*ones(N,1);                                          % Relabeled degree vector
     based on y_2 (eq. 10, page 11)
N_rem_1 = ceil(nnz(A)/2*trsh/100);                              % Number of links, whose
    weight to scale wihtin one iteration
Dist_M_1 = (abs(diag(1:N)*A_n - A_n*diag(1:N))).*A_n;           % Compute the distance
    matrix between any two adjacent nodes (sec 5.1, page 15)
% Dist_M_1 = (ones(N) + abs(diag(1:N)*A_n - A_n*diag(1:N))).*A_n.*C;  % Compute the distance
    matrix between any two adjacent nodes (sec 5.1, page 15)
A_new = A_n;                                                    % Store the adjacency
    matrix
counter_1 = 1;                                                  % Coutner for the number
    of links with scaled weight
while counter_1 < N_rem_1                                       % Run the loop until
    N_rem links are scaled in weight
    tr = 1;                                                     % Initialise the
        indicator function in case the link is found.
    while (tr)                                                  % Run the loop until the
        link is removed
        Dist_v = Dist_M_1(:); Dist_v = Dist_v(Dist_v > 0);
        val = max(Dist_v);                                  % Find the maximum distance
            value
        %         val = max(Dist_M_1(:));                                % Find the
            maximum distance value
        [ind_1,ind_2] = find(Dist_M_1 == val(1));                % Identify the link
            connecting two most far away nodes
        if((Deg_v(ind_1(1))) > 1 && (Deg_v(ind_2(1))) > 1 ...
                && A_new(ind_1(1),ind_2(1)) == 1)                % Check the validity
                    of a link (i.e. if adjacent nodes have degree > 1)?
            Dist_M_1(ind_1(1),ind_2(1)) = 0;                    % Define the distance of
                a chosen link as 0, so it is not conisdered anymore
            Dist_M_1(ind_2(1),ind_1(1)) = 0;
            counter_1 = counter_1 + 1; tr = 0;                  % Increment the counter
            Deg_v(ind_1(1)) = Deg_v(ind_1(1)) - 1;              % Update the degree
                values for each node
            Deg_v(ind_2(1)) = Deg_v(ind_2(1)) - 1;
            A_new(ind_1(1),ind_2(1)) = (0.001/(N_pr))*br_it;    % Scale the weight of a
                link
            A_new(ind_2(1),ind_1(1)) = (0.001/(N_pr))*br_it;
        else                                                    % This link can not be
            removed
            Dist_M_1(ind_1(1),ind_2(1)) = 0;                    % Define the distance of
                a chosen link as 0, so it is not conisdered anymore
            Dist_M_1(ind_2(1),ind_1(1)) = 0;
        end
    end
end
A_new(ind,ind) = A_new;
A_rem = (A - A_new);
Deg_new = A_new*ones(N,1);
% Check if there are zero-degree nodes (testing the code)
```

```
123 if(nnz(Deg_new) < N)
124     disp('Number of zero degree nodes:')
125     disp(length(Deg_new) - nnz(Deg_new))
126     plot(sort(Deg_new))
127     error('Something is wrong!!!')
128 end
129 end
130
131 function [ind_rank,pos_rank,y_2] = compute_y_2(W)
132 [Y,B] = eig(W);                     % Compute the eigenvalue decomposition of I + W - diag(W*
        u)
133 [~,ind_pe] = maxk(diag(B),3);       % Identify the three largest eigenvalues
134 y_2 = Y(:,ind_pe(2));               % Store the eigenvector y_2
135 [pos_rank,ind_rank] = sort(y_2);    % Compute position and ranking of eacn node in y_2
136 end
137
138 function W = compute_W(A,N,alpha,delta)
139 A_s = zeros(N);                                                                    %
        Initialise A.*A^2
140 for i = 1 : N                                                                      %
        Compute A.*A^2 as in algorithm 2 (page 17)
141     ind_n = find(A(i,:) == 1);                                                     % (
            pseudocode 2, line 3)
142     for j = 1  : length(ind_n)                                                     % (
            pseudocode 2, line 3)
143         ind_n_2 = find(A(ind_n(j),:) == 1); ind_p_2 = ind_n_2(ind_n_2 > i);        % (
                pseudocode 2, line 4)
144         A_s(i,ind_p_2) = A_s(i,ind_p_2) + A(i,ind_p_2).*A(ind_n(j),ind_p_2);       % (
                pseudocode 2, line 5)
145     end
146 end
147 A_s = A_s + A_s';                                                                  %
        Compute A.*A^2 (pseudocode 2, line 9)
148 Deg_inv = diag(A*ones(N,1))^-1;                                                    %
        Compute vector with inverse degrees of each node
149 W = (alpha + delta)*Deg_inv*(A_s + A)*Deg_inv + 0.5*delta*(Deg_inv*A + A*Deg_inv);  %
        Compute matrix W (Theorem 1, page 6)
150 end
151
152 function [M,m,C_out,N_clusters] = optimize_m(A)
153 % This function computes merger matrix M, number of clusters c and cluster
154 % membership function C
155 N = size(A,1);                                                  % Number of nodes N
156 Deg = A*ones(N,1);                                              % Degree vector
157 L_2 = nnz(A);                                                   % Twice number of links
158 C = ident_clusters_border(A,N,Deg,L_2,0);                       % Identify cluster
        membership of each node
159 if(isempty(C))                                                  % If there are no
        clusters
160     M = 0;m = 0;C_out=ones(N,1);N_clusters = 1;return           % Return only one cluster
            , with all nodes in it
161 end
162 A_mat = A - (1/nnz(A)).*(Deg*Deg');                             % Compute the modularity
        matrix A - 1/2L*d*d^T
163 [M,C_out,N_clusters] = compute_M(C,A_mat,A,N);                 % Compute the modularity
        matrix M
164 [M,C_out,N_clusters] = merge_clusters(M,C_out,N_clusters);     % optimize partition by
        maximising modularity
165 m = compute_modularity_m(A,C_out);                            % Compute the modularity
        for a given partition
166 end
167
168 function C = ident_clusters_border(A,N,Deg,L_2,trsh)
169 % This function implements the algorithm 1 from the LCP paper (page 13)
170 Mod_1 = zeros(N,1);         Mod_2 = zeros(N,1);                                    %
         Initialise the modularity vectors (pseudocode 1, line 2)
171 Mod_1(1) = -Deg(1)^2/(L_2); Mod_2(N) = -Deg(N)^2/(L_2);                            %
         Store the first value of the modularity vectors (pseudocode 1, line 4-5)
172 A_mat = A - (Deg*Deg')./L_2;
173 for br = 2 : N                                                                    %
         Iteratively compute the modularity of each possible partition (pseudocode 1, lines 7 -
```

```
    15)
174     Mod_1(br) = Mod_1(br-1) + 2*sum(A_mat(1:br-1,br)) + A_mat(br,br);                    %
            Update the modularity of each possible bisection (pseudocode 1, line 9,11,13)
175     Mod_2(N-br+1) = Mod_2(N-br+2) + 2*sum(A_mat(N-br+2:N,N-br+1)) + A_mat(N-br+1,N-br+1);   %
            Update the modularity of each possible bisection (pseudocode 1, line 10,12,14)
176 end
177 [m_max,ind_max] = max(Mod_1+Mod_2);                                                      %
        Determine the best partition into two clusters       (pseudocode 1, line 16)
178 m_max = m_max(1); ind_max = ind_max(1);                                                 %
        Store the cluster border and the obtained modularity (pseudocode 1, line 16)
179 if(m_max >= trsh && ind_max > 1 && ind_max < N)                                         %
        If the new partition improves modularity of the parent cluster, adopt it (pseudocode 1,
    line 17)
180     A_1 = A(1:ind_max,1:ind_max); Deg_1 = Deg(1:ind_max); N_1 = ind_max;                 %
            Perform the partition and compute adjacency matrix and degree vector for the first
        cluster (pseudocode 1, line 18)
181     A_2 = A(ind_max+1:N,ind_max+1:N); Deg_2 = Deg(ind_max+1:N); N_2 = N - ind_max;       %
            Perform the partition and compute adjacency matrix and degree vector for the second
        cluster (pseudocode 1, line 18)
182     C = [ident_clusters_border(A_1,N_1,Deg_1,L_2,Mod_1(ind_max)),ind_max,...
183         ind_max + ident_clusters_border(A_2,N_2,Deg_2,L_2,Mod_2(ind_max))];               %
                Call the function to check if the obtained two clusters can be further
                partitioned (pseudocode 1, line 20)
184 else                                                                                    %
        otherwise, adopt only the parent cluster (pseudocode 1, line 21)
185     C = [];
186 end
187 end
188
189 function [M,C_vec,N_cl] = compute_M(C,A_mat,A,N)
190 % This function computes the c x c modularity matrix M and the Nx1 cluster
191 % membership vector C
192 Inv = 1/nnz(A);                                     % 1/2L
193 N_cl = length(C)+1;                                 % Number of clusters
194 C_low = [1,C]; C_upp = [C,N]; C_vec = zeros(N,1);   % Explanation
195 for br = 1 : N_cl                                   % Construct cluster membership vector C
196     C_vec(C_low(br):C_upp(br)) = br;
197 end
198 M = zeros(N_cl);                                    % Construct the cluster membership matrix
        M
199 for br_1 = 1 : N_cl
200     for br_2 = 1 : br_1
201         M(br_1,br_2) = Inv*sum(sum(A_mat(C_low(br_1):C_upp(br_1),C_low(br_2):C_upp(br_2))));
202         M(br_2,br_1) = M(br_1,br_2);
203     end
204 end
205 end
206
207 function [M,C_vec,N_cl] = merge_clusters(M,C_vec,N_cl)
208 % This function further optimizes the modularity m
209 Off_diag_M = diag(M,1);                                                  % Store the elements
        above the main diagonal of the modularity matrix M
210 while((max(Off_diag_M) > 0))                                             % While there is a
        positive off-diagonal element in M
211     ind = find(Off_diag_M == max(Off_diag_M));                           % Find position of the
            largest off-diagonal element in M
212     for br = ind + 1 : N_cl                                              % Since we merge two
            clusters, update memebership of all nodes affected
213         C_vec(C_vec == br) = br - 1;                                     % Update the cluster
                membership vector C
214     end
215     N_cl = N_cl - 1;                                                     % Update number of
            clusters
216     M(:,ind) = M(:,ind) + M(:,ind+1); M(:,ind+1) = [];                   % Update the cluster
            membership matrix M
217     M(ind,:) = M(ind,:) + M(ind+1,:); M(ind+1,:) = [];                   % Update the cluster
            membership matrix M
218     Off_diag_M = diag(M,1);                                              % Update the off-diagonal
            elements of M
219 end
220 end
```

```
221
222 function Q = compute_modularity_m(A,C)
223 N = size(A,1); Deg = A*ones(N,1);
224 Q = 1/nnz(A).*sum(sum((A - (1./nnz(A).*Deg*Deg')).*((ones(N,1)*C' - C*ones(1,N)) == 0)));
225 end
226
227 function [ind_rank,pos_rank,y_2] = compute_y_2_v2(A_model)
228 N = size(A_model,1);
229 A_m = A_model - 1/N*ones(N);
230 Vec = rand(N,1);
231 ind = 1;
232 cnt = 1;
233 while(ind)
234     cnt = cnt + 1;
235     Vec_est = A_m*Vec;
236     Vec_est = Vec_est./(sqrt(sum(Vec_est.*Vec_est)));
237     if(mean(abs(Vec - Vec_est)) < 1e-5)
238         ind = 0;
239     else
240         Vec = Vec_est;
241     end
242 end
243 [pos_rank,ind_rank] = sort(Vec);    % Compute position and ranking of eacn node in y_2
244 y_2 = Vec;
245 end
```

B.1.2. LCP for a known number of communities

```
1  function [C,c,m] = LCP_c(A,c)
2  % This function implements the Linear Clustering Process (LCP), from the
3  % publication
4  N = size(A,1);                                                          %
       Number of nodes
5  N_pr = 40;                                                              %
       Number of iterations
6  L_per = 60/N_pr;                                                        %
       Ratio of links whose weight will be scaled
7  L_rem = linspace(L_per,L_per,N_pr);                                     %
       Scaling intensity for links in each iteration
8
9  alpha = 0.95;                                                           %
       Attraction strength
10 delta = 1e-3;                                                           %
       Repulsion strength
11
12 A_fm = A;                                                               %
       Store the adjacency matrix
13
14 Inds = zeros(N,N_pr);                                                   %
       Initalise the ranking of each node in y_2 per iteration
15 Pos_M = zeros(N,N_pr);                                                  %
       Initalise the position of each node in y_2 per iteration
16 m_it = zeros(N_pr,1);                                                   %
       Initalise the modularity in each iteration
17 C_it = zeros(N,N_pr);                                                   %
       Initalise the number of clusters in each iteration
18 c_it = zeros(N_pr,1);                                                   %
       Initalise the modularity in each iteration
19
20 W = compute_W(A_fm,N,alpha,delta);                                      %
       Compute W matrix (we compute it only once!)
21
22 counter = 1;
23 while(counter <= N_pr)                                                  %
       Perform N_pr iterations of LCP
24     A_model = eye(N) + A_fm.*W - diag(A_fm.*W*ones(N,1));               %
           Compute state space matrix                                  % Compute the
           position vector in the steady state
25     [ind_r,pos_r,~] = compute_y_2_v2(A_model);                          %
           Compute the position vector in the steady state
```

```
26      Inds(:,counter) = ind_r; Pos_M(:,counter) = pos_r;                                  %
            Store the position of each node per iteration
27      [~,m_it(counter),C_it(:,counter),c_it(counter)] = ident_communities_v1_N(A(ind_r,ind_r),c
            );   % Identify the clusters, for given position vector
28      [A_fm,~,~] = scale_links(A_fm,ind_r,L_rem(counter),counter,N_pr);                   %
            Scale the weights of the identified inter-community links links and recompute the
            adjacency matrix
29      counter = counter + 1;                                                              %
            Update the counter
30  end
31  [val_mod,ind] = max(m_it);                                                              %
        Adopt the iteration with the highest modularity
32  c = c_it(ind);                                                                          %
        Output the estimated number of clusters
33  m = val_mod;                                                                            %
        Output the estimated modularity
34  C = zeros(N,1);
35  C(Inds(:,ind)) = C_it(:,ind);                                                           %
        Output the cluster membership function of each node
                                                                                        % Return C
        and m
36  end
37
38  %% Used sub-functions
39  function W = compute_W(A,N,alpha,delta)
40  A_s = zeros(N);                                                                         %
        Initialise A.*A^2
41  for i = 1 : N                                                                           %
        Compute A.*A^2 as in algorithm 2 (page 17)
42      ind_n = find(A(i,:) == 1);                                                          % (
            pseudocode 2, line 3)
43      for j = 1  : length(ind_n)                                                          % (
            pseudocode 2, line 3)
44          ind_n_2 = find(A(ind_n(j),:) == 1); ind_p_2 = ind_n_2(ind_n_2 > i);             % (
                pseudocode 2, line 4)
45          A_s(i,ind_p_2) = A_s(i,ind_p_2) + A(i,ind_p_2).*A(ind_n(j),ind_p_2);            % (
                pseudocode 2, line 5)
46      end
47  end
48  A_s = A_s + A_s';                                                                       %
        Compute A.*A^2 (pseudocode 2, line 9)
49  Deg_inv = diag(A*ones(N,1))^-1;                                                         %
        Compute vector with inverse degrees of each node
50  W = (alpha + delta)*Deg_inv*(A_s + A)*Deg_inv - 0.5*delta*(Deg_inv*A + A*Deg_inv);      %
        Compute matrix W (Theorem 1, page 6)
51  % W = (alpha + 1/(1+alpha)^4)*Deg_inv*(A_s + A)*Deg_inv - 0.5*1/(1+alpha)^4*(Deg_inv*A + A*
        Deg_inv);      % Compute matrix W (Theorem 1, page 6)
52  end
53
54  function [ind_rank,pos_rank,y_2] = compute_y_2_v2(A_model)
55  N = size(A_model,1);
56  A_m = A_model - 1/N*ones(N);
57  Vec = rand(N,1);
58  ind = 1;
59  cnt = 1;
60  while(ind)
61      cnt = cnt + 1;
62      Vec_est = A_m*Vec;
63      Vec_est = Vec_est./(sqrt(sum(Vec_est.*Vec_est)));
64      if(mean(abs(Vec - Vec_est)) < 1e-5)
65          ind = 0;
66      else
67          Vec = Vec_est;
68      end
69  end
70  [pos_rank,ind_rank] = sort(Vec);    % Compute position and ranking of eacn node in y_2
71  y_2 = Vec;
72  end
73
74  function [A_new,A_rem,Deg_new] = scale_links(A,ind,trsh,br_it,N_pr)
75  % Adjacency matrix A provided as input is not relabeled!
```

```
N = size(A,1);                                                  % Network size
A_n = A(ind,ind);                                               % Relabeled adjacency
    matrix A based on y_2 (eq. 10, page 11)
Deg_v = A_n*ones(N,1);                                          % Relabeled degree vector
     based on y_2 (eq. 10, page 11)
N_rem_1 = ceil(nnz(A)/2*trsh/100);                              % Number of links, whose
    weight to scale wihtin one iteration
Dist_M_1 = (abs(diag(1:N)*A_n - A_n*diag(1:N))).*A_n;           % Compute the distance
    matrix between any two adjacent nodes (sec 5.1, page 15)
A_new = A_n;                                                    % Store the adjacency
    matrix
counter_1 = 1;                                                  % Coutner for the number
    of links with scaled weight
while counter_1 < N_rem_1                                       % Run the loop until
    N_rem links are scaled in weight
    tr = 1;                                                     % Initialise the
        indicator function in case the link is found.
    while (tr)                                                  % Run the loop until the
        link is removed
        val = max(Dist_M_1(:));                                 % Find the maximum
            distance value
        [ind_1,ind_2] = find(Dist_M_1 == val);                  % Identify the link
            connecting two most far away nodes
        if((Deg_v(ind_1(1))) > 1 && (Deg_v(ind_2(1))) > 1 ...
            && A_new(ind_1(1),ind_2(1)) == 1)                   % Check the validity of a
                link (i.e. if adjacent nodes have degree > 1)?
            Dist_M_1(ind_1(1),ind_2(1)) = 0;                    % Define the distance of
                a chosen link as 0, so it is not conisdered anymore
            Dist_M_1(ind_2(1),ind_1(1)) = 0;
            counter_1 = counter_1 + 1; tr = 0;                  % Increment the counter
            Deg_v(ind_1(1)) = Deg_v(ind_1(1)) - 1;              % Update the degree
                values for each node
            Deg_v(ind_2(1)) = Deg_v(ind_2(1)) - 1;
            A_new(ind_1(1),ind_2(1)) = (0.01/(N_pr))*br_it;     % Scale the weight of a
                link
            A_new(ind_2(1),ind_1(1)) = (0.01/(N_pr))*br_it;
        else                                                    % This link can not be
            removed
            Dist_M_1(ind_1(1),ind_2(1)) = 0;                    % Define the distance of
                a chosen link as 0, so it is not conisdered anymore
            Dist_M_1(ind_2(1),ind_1(1)) = 0;
        end
    end
end
A_new(ind,ind) = A_new;
A_rem = (A - A_new);
Deg_new = A_new*ones(N,1);
% Check if there are zero-degree nodes (testing the code)
if(nnz(Deg_new) < N)
    disp('Number of zero degree nodes:')
    disp(length(Deg_new) - nnz(Deg_new))
    plot(sort(Deg_new))
    error('Something is wrong!!!')
end
end

function [M,m,C_out,N_clusters] = ident_communities_v1_N(A,N_cl)
% Estimate communities in the network
N = size(A,1);
Deg = A*ones(N,1);
% Step 1: Find cluster borders
L_2 = nnz(A);
C = ident_clusters_v2_N(A,N,Deg,L_2,1,ceil(log2(N_cl))+1);
if(isempty(C))
    M = 0;m = 0;C_out=ones(N,1);N_clusters = 1;return
end
A_mat = A - (1/nnz(A)).*(Deg*Deg');
%% Step 2: Compute the modularity matrix M
[M,C_out,N_clusters] = compute_modularity_matrix_v1(C,A_mat,A,N);
%% Step 3: optimize partition by maximising modularity
[M,C_out,N_clusters] = optimize_clusters_v1_N(M,C_out,N_clusters,N_cl);
```

```
% m = trace(M)
m = compute_modularity_v1(A,C_out);
end

function C = ident_clusters_v2_N(A,N,Deg,L_2,br_it,N_it)
Mod_1 = zeros(N,1);          Mod_2 = zeros(N,1);
Mod_1(1) = -Deg(1)^2/(L_2);   Mod_2(N) = -Deg(N)^2/(L_2);
A_mat = A - (Deg*Deg')./L_2;
for br = 2 : N
    Mod_1(br) = Mod_1(br-1) + 2*sum(A_mat(1:br-1,br)) + A_mat(br,br);
    Mod_2(N-br+1) = Mod_2(N-br+2) + 2*sum(A_mat(N-br+2:N,N-br+1)) + A_mat(N-br+1,N-br+1);
end
[m_max,ind_max] = max(Mod_1+Mod_2);       % Determine cluster border
m_max = m_max(1); ind_max = ind_max(1);
if((ind_max == 1 || ind_max == N) && br_it < N_it && N > 2)
    ind_max = ceil(N/2);
    A_1 = A(1:ind_max,1:ind_max); Deg_1 = Deg(1:ind_max); N_1 = ind_max;
    A_2 = A(ind_max+1:N,ind_max+1:N); Deg_2 = Deg(ind_max+1:N); N_2 = N - ind_max;
    C = [ident_clusters_v2_N(A_1,N_1,Deg_1,L_2,br_it + 1,N_it),ind_max,...
         ind_max + ident_clusters_v2_N(A_2,N_2,Deg_2,L_2,br_it + 1,N_it)];
elseif(ind_max > 1 && ind_max < N && br_it < N_it && N > 2)% Is there a new cluster border?
    A_1 = A(1:ind_max,1:ind_max); Deg_1 = Deg(1:ind_max); N_1 = ind_max;
    A_2 = A(ind_max+1:N,ind_max+1:N); Deg_2 = Deg(ind_max+1:N); N_2 = N - ind_max;
    C = [ident_clusters_v2_N(A_1,N_1,Deg_1,L_2,br_it + 1,N_it),ind_max,...
         ind_max + ident_clusters_v2_N(A_2,N_2,Deg_2,L_2,br_it + 1,N_it)];
else
    C = [];
end;end

function [M,C_vec,N_cl] = compute_modularity_matrix_v1(C,A_mat,A,N)
Inv = 1/nnz(A);     % 1/2L
N_cl = length(C)+1; % Number of clusters
C_low = [1,C]; C_upp = [C,N]; C_vec = zeros(N,1);
for br = 1 : N_cl   % Construct cluster membership vector C
    C_vec(C_low(br):C_upp(br)) = br;
end
M = zeros(N_cl);    % Construct the cluster membership matrix M
for br_1 = 1 : N_cl
    for br_2 = 1 : br_1
        M(br_1,br_2) = Inv*sum(sum(A_mat(C_low(br_1):C_upp(br_1),C_low(br_2):C_upp(br_2))));
        M(br_2,br_1) = M(br_1,br_2);
    end
end
end

function [M,C_vec,N_cl] = optimize_clusters_v1_N(M,C_vec,N_cl,c)
Test = diag(M,1);
while(length(Test) > c-1)                                          % Find position of the
    largest positive off diagonal element of M
    ind = find(Test == max(Test));
    for br = ind + 1 : N_cl                                        % Update the cluster
        membership vector
        C_vec(C_vec == br) = br - 1;
    end
    N_cl = N_cl - 1;                                               % Update the number of
        clusters
    M(:,ind) = M(:,ind) + M(:,ind+1); M(:,ind+1) = [];             % Update the cluster
        membership matrix M
    M(ind,:) = M(ind,:) + M(ind+1,:); M(ind+1,:) = [];
    Test = diag(M,1);                                              % Update the second
        diagonal vector
end;end

function Q = compute_modularity_v1(A,C)
N = size(A,1);
Deg = A*ones(N,1);
Q = 1/nnz(A).*sum(sum((A - (1./nnz(A).*Deg*Deg')).*((ones(N,1)*abs(C)' - C*ones(1,N)) == 0)))
    ;
end
```

B.1.3. Non-backtracking variant of LCP

```
function [Comm,N_cl,Q] = Non_back_tracking_LCP(A,N)                                    % LCP-based non back tracking approach for estimating number of clusters (Section 4.4)
Deg = A*ones(N,1);                                                                     % Degree vector
gamma = fit_powerlaw(Deg); %fit power-law exponent gamma
if gamma <= 1 %determine if the network conform power-law
    alpha = 0.95;
elseif gamma <= 3.5
    alpha = 0.3*gamma - 0.1;
else
    alpha = 0.95;
end

H = [eye(N) + alpha*(A.*A^2+A) - diag(alpha*(A.*A^2+A)*ones(N,1)) - (eye(N) - diag(Deg)), (eye(N) - diag(Deg)); eye(N), zeros(N)];  % The 2Nx2N matrix in eq. (28), page 15
Lambda_H = eigs(H,2*N);                                                                % Compute the eigenvalues of H
eig_max = maxk(real(Lambda_H),2);                                                      % Determine the largest eigenvalue
N_cl = nnz(intersect(find(real(Lambda_H) > sqrt(eig_max(1))),find(imag(Lambda_H) == 0)));   % Determine the number of real eigenvalues larger that sqrt(lambda_1)
[U,~] = eigs(H,N_cl,'la');
U = real(U);
Comm = kmeans(U(1:N,:),N_cl);
Q = compute_modularity(A,Comm);
end

function Q = compute_modularity(A,C)
N = size(A,1);
Deg = A*ones(N,1);
Q = 1/nnz(A).*sum(sum((A - (1./nnz(A).*Deg*Deg')).*((ones(N,1)*abs(C)' - C*ones(1,N)) == 0)));
end

function gamma = fit_powerlaw(degree)
prob = tabulate(degree);
prob(:,2) = [];
prob_0 = prob(:,2) == 0;
prob(prob_0,:) = [];
prob = log(prob);
gamma = -polyfit(prob(:,1),prob(:,2),1)*[1 0]';
end
```

B.2. Newman Method

```
function [C_out,N_cl,m] = Newman(A)
N = size(A,1);
Deg = A*ones(N,1);
M = A - (Deg*Deg')./nnz(A);
C = ones(N,1);
C_out = Newman_ident_cl(C,M,0);
m = compute_modularity(A,C_out);
uniComm = unique(C_out);
for i=1:length(uniComm)
    C_out(C_out==uniComm(i)) = max(uniComm) +i;
end
uniComm = unique(C_out);
for i=1:length(uniComm)
    C_out(C_out==uniComm(i)) = i;
end
N_cl = max(C_out);
end

function C_out = Newman_ident_cl(C,M,tr)
C_out = C;
[X,L] = eig(M);
[val,ind] = max(diag(L));
```

```
ind_pos = find(X(:,ind) > 0);
ind_neg = find(X(:,ind) < 0);

if(val <= 0 || isempty(ind_pos) || isempty(ind_neg))
    return
else
    C_out(ind_pos) = C(ind_pos).*rand();
    C_out(ind_neg) = C(ind_neg).*rand();

    M_pos = M(ind_pos,ind_pos);
    M_neg = M(ind_neg,ind_neg);

    m_pos = sum(sum(M_pos));
    m_neg = sum(sum(M_neg));

    if(m_pos + m_neg <= tr)
        return
    else
        M_pos = M_pos - diag(M_pos*ones(length(ind_pos),1));
        M_neg = M_neg - diag(M_neg*ones(length(ind_neg),1));

%         C_out(ind_pos) = Newman_ident_cl(C_out(ind_pos),M_pos,m_pos);
%         C_out(ind_neg) = Newman_ident_cl(C_out(ind_neg),M_neg,m_neg);
        C_out(ind_pos) = Newman_ident_cl(C_out(ind_pos),M_pos,0);
        C_out(ind_neg) = Newman_ident_cl(C_out(ind_neg),M_neg,0);
    end
end
end

function Q = compute_modularity(A,C)
N = size(A,1);
Deg = A*ones(N,1);
Q = 1/nnz(A).*sum(sum((A - (1./nnz(A).*Deg*Deg')).*((ones(N,1)*abs(C)' - C*ones(1,N)) == 0)))
    ;
end
```

B.3. Louvain Method

```
function [C,c,Q] = Louvain(A)
A_0 = A;                                % Store the original adjacency matrix
N = size(A,1);                          % Number of nodes
P = 1:N;                                % Each node is a community
Q = compute_modularity(A_0,P);          % Compute modularity
done = 0;                               % Indicator for stopping the code
it = 1;                                 % Initialise the iteration counter

while(~done)                            % Until modulairty m cannot be improved further
    P = MoveNodes(A,P,Q);               % Move nodes to the best community
    P = simplify_C(P);                  % Define partition from 1 to c
    C_st{1,it} = P;                     % Store the current partition before aggregaton
    Q = compute_modularity(A,P);        % Compute modularity
    done = (length(unique(P)) == N);    % Check if there were no changes
    if(~done)                           % In case there were changes
        [A,P,N] = AggregateGraph(A,P);  % Aggregate the graph in c nodes
        Q = compute_modularity(A,P);    % Compute modularity
        it = it + 1;                    % Increment the iteration counter
    end
end
C = compute_partition(C_st,size(A_0,1));% Compute the final partition
c = length(unique(C));                  % Compute the number of clusters c
Q = compute_modularity(A_0,C);          % Compute modularity
end

function P = MoveNodes(A,P,m_old)
N = size(A,1);
m = m_old;
Deg = A*ones(N,1);
Q_p = 1/nnz(A).*(A - (1./nnz(A).*Deg*Deg'));
while(m >= m_old)
```

```
    nodes = randperm(N);                                        % Visit nodes at random
    L_2 = sum(sum(A));                                          % Sum of all link weights
    for counter = 1 : N                                         % Go over each node in the
        graph
        ind_neig = intersect(find(P~=P(nodes(counter))),find(A(:,nodes(counter))));        %
            Find neighbours of node i
        Delta_m = zeros(length(ind_neig),1);                    % Initialise the delta_m
            vector of node i
        for counter_neig = 1 : length(ind_neig)                 % For each nieghbour j of
            node i
            ind_comm = find(P == P(ind_neig(counter_neig)));    % Determine all nodes from
                the community of node j
            Sum_in = sum(sum(A(ind_comm,ind_comm)));            % Sum of links within the
                community C
            Sum_tot = 2*sum(sum(A(ind_comm,:)));                % Sum of the weights of links
                 incident to nodes in community C
            Sum_C = sum(A(ind_comm,nodes(counter)));            % Sum of the weights of links
                 from node i to nodes in community C
            d_i = sum(A(:,nodes(counter)));                     % Sum of the link weights
                adjacent to node i
            Delta_m(counter_neig) = ((Sum_in + 2*Sum_C)/L_2 - ((Sum_tot + d_i)/L_2)^2) - (
                Sum_in/L_2 - (Sum_tot/L_2)^2 - (d_i/L_2)^2);   % Compute the modularity gain
        end
        [ind_val,ind_pos] = max(Delta_m);                       % Determine the best
            community for node i
        if(ind_val > 0)                                         % If the modularity can be
            improved?
            ind_sub_1 = find(P == P(nodes(counter))); ind_sub_1(ind_sub_1~=nodes(counter));
            P(nodes(counter)) = P(ind_neig(ind_pos));        % Update the cluster membership
                of each node
            ind_sub_2 = find(P == P(ind_neig(ind_pos))); ind_sub_2(ind_sub_2~=nodes(counter))
                ;
            m = m - 2*sum(Q_p(nodes(counter),ind_sub_1)) + 2*sum(Q_p(nodes(counter),ind_sub_2
                ));
        end
    end
    if(m <= m_old)                                              % If the modularity is not
        improved
        return                                                  % Break the while loop
    end
    m_old = m;                                                  % Update the old modularity
end
end

function [A_new,P_new,N_new] = AggregateGraph(A,P) % Aggregate the graph
N_new = length(P);
A_new = zeros(N_new);
P_new = 1 : N_new;
for counter_i = 1 : N_new
    for counter_j = 1 : N_new
        if(counter_i == counter_j)
            ind_comm_i = find(P == counter_i);
            A_new(counter_i,counter_i) = 0.5*sum(sum(A(ind_comm_i,ind_comm_i)));
        else
            ind_comm_i = (P == counter_i);
            ind_comm_j = (P == counter_j);
            A_new(counter_i,counter_j) = sum(sum(A(ind_comm_i,ind_comm_j)));
        end
    end
end
end

function Q = compute_modularity(A,C) % Compute modularity of the given partition
N = size(A,1);
Deg = A*ones(N,1);
Q = 1/nnz(A).*sum(sum((A - (1./nnz(A).*Deg*Deg')).*((ones(N,1)*C - C'*ones(1,N)) == 0)));
end

function  P = simplify_C(P) % Define clusters from 1 to c
P_un = unique(P);
for counter = 1 : length(P_un)
```

```
    ind = (P == P_un(counter));
    P(ind) = counter;
end
end

function C = compute_partition(C_st,N) % Restore the partition from C_st
it = length(C_st);
C = zeros(1,N);
for counter_1 = 1 : N
    C(counter_1) = C_st{1,1}(counter_1);
    for counter_2 = 2 : it
        C(counter_1) = C_st{1,counter_2}(C(counter_1));
    end
end
end
```

B.4. Leiden Method

```
function [Comm,C,Q] = Leiden(A)
A_0 = A;                                % Store the original adjacency matrix
N = size(A,1);                          % Number of nodes
P = 1:N;                                % Each node is a community
done = 0;                               % Indicator for stopping the code
it = 1;                                 % Initialise the iteration counter

while ~done
    P = MoveNodesFast(A,P);
    P_refined = RefinePartition(A,P);
    P = simplify_C(P_refined);                  % Define partition from 1 to c
    C_st{1,it} = P;                     % Store the current partition before aggregaton
    done = (length(unique(P)) == N);    % Check if there were no changes
    if(~done)                           % In case there were changes
        [A,P,N] = AggregateGraph(A,P);  % Aggregate the graph in c nodes
        it = it + 1;                    % Increment the iteration counter
    end
end
Comm = compute_partition(C_st,size(A_0,1));% Compute the final partition
C = length(unique(Comm));                   % Compute the number of clusters c
Q = compute_modularity(A_0,Comm);           % Compute modularity
end

function P = MoveNodesFast(A,P)
N = size(A,1);
queue = randperm(N);                                        % Visit nodes at random
L_2 = sum(sum(A));                                          % Sum of all link weights
while ~isempty(queue)
    node_v = queue(1);
    queue(1) = [];
    ind_neig = find(A(:,node_v));        % Find neighbours of node v
    Delta_m = zeros(length(ind_neig),1);                    % Initialise the delta_m vector
        of node v
    for counter_neig = 1 : length(ind_neig)                 % For each nieghbour j of node v
        ind_comm = find(P == P(ind_neig(counter_neig)));    % Determine all nodes from the
            community of node j
        Sum_in = sum(sum(A(ind_comm,ind_comm)));            % Sum of links within the
            community C
        Sum_tot = 2*sum(sum(A(ind_comm,:)));                % Sum of the weights of links
            incident to nodes in community C
        Sum_C = sum(A(ind_comm,node_v));            % Sum of the weights of links from node v
             to nodes in community C
        d_i = sum(A(:,node_v));                     % Sum of the link weights adjacent to
            node v
        Delta_m(counter_neig) = ((Sum_in + 2*Sum_C)/L_2 - ((Sum_tot + d_i)/L_2)^2) - (Sum_in/
            L_2 - (Sum_tot/L_2)^2 - (d_i/L_2)^2);   % Compute the modularity gain
    end
    [ind_val,ind_pos] = max(Delta_m);                       % Determine the best community
        for node i
    if ind_val > 0
        P(node_v) = P(ind_neig(ind_pos));       % Update the cluster membership of each node
```

```
    end
end
end

function P_refined = RefinePartition(A,P)
N = size(A,1);
P_refined = 1:N;
C = unique(P);
for i = 1:length(C)
    S = find(P == C(i));
    P_refined = MergeNodesSubset(A,P_refined,S,C(i));
end

end

function P = MergeNodesSubset(A,P,S,C)
R = [];
gamma = 1/7;
for i = 1:length(S)
    S_v = setdiff(S,S(i));
    recur_v = sum(A(:,S(i)));
    recur_S = length(S);
    E_v_Sv = length(intersect(find(A(:,S(i))), S_v));
    if E_v_Sv >= gamma * recur_v * (recur_S - recur_v)
        R = [R S(i)];
    else
        P(S(i)) = C;
    end
end

theta = 1;
for i = 1:length(R)
    T = [];
    if R(i)==P(R(i))
        for j = 1:length(S)
            S_C = setdiff(S,S(j));
            recur_C = sum(A(:,S(j)));
            recur_S = length(S);
            E_C_SC = length(intersect(find(A(:,S(j))), S_C));
            if E_C_SC >= gamma * recur_C * (recur_S - recur_C)
                T = [T S(j)];
            end
        end
        L_2 = sum(sum(A));
        Delta_m = zeros(length(T),1);                    % Initialise the delta_m vector of
            node v
        for counter_T = 1 : length(T)                % For each nieghbour j of node v
            ind_comm = find(P == P(T(counter_T)));    % Determine all nodes from the
                community of node j
            Sum_in = sum(sum(A(ind_comm,ind_comm)));            % Sum of links within the
                community C
            Sum_tot = 2*sum(sum(A(ind_comm,:)));                % Sum of the weights of links
                 incident to nodes in community C
            Sum_C = sum(A(ind_comm,R(i)));            % Sum of the weights of links from node
                 v to nodes in community C
            d_i = sum(A(:,R(i)));                     % Sum of the link weights adjacent to
                node v
            Delta_m(counter_T) = ((Sum_in + 2*Sum_C)/L_2 - ((Sum_tot + d_i)/L_2)^2) - ...
                (Sum_in/L_2 - (Sum_tot/L_2)^2 - (d_i/L_2)^2);   % Compute the modularity gain
        end
        if Delta_m > 0
            Pr_C = exp((1/theta) * Delta_m);
            Pr_C = Pr_C/sum(Pr_C);
            Random = randsrc(1,1,[1:length(Pr_C); Pr_C']);
            P(R(i)) = T(Random);
        else
            P(R(i)) = C;
        end
    else
        P(R(i)) = C;
    end
```

```
end
end

function Q = compute_modularity(A,C) % Compute modularity of the given partition
N = size(A,1);
Deg = A*ones(N,1);
Q = 1/nnz(A).*sum(sum((A - (1./nnz(A).*Deg*Deg')).*((ones(N,1)*C - C'*ones(1,N)) == 0)));
end

function [A_new,P_new,N_new] = AggregateGraph(A,P) % Aggregate the graph
N_new = length(P);
A_new = zeros(N_new);
P_new = 1 : N_new;
for counter_i = 1 : N_new
    for counter_j = 1 : N_new
        if(counter_i == counter_j)
            ind_comm_i = find(P == counter_i);
            A_new(counter_i,counter_i) = 0.5*sum(sum(A(ind_comm_i,ind_comm_i)));
        else
            ind_comm_i = (P == counter_i);
            ind_comm_j = (P == counter_j);
            A_new(counter_i,counter_j) = sum(sum(A(ind_comm_i,ind_comm_j)));
        end
    end
end
end

function  P = simplify_C(P) % Define clusters from 1 to c
P_un = unique(P);
for counter = 1 : length(P_un)
    ind = (P == P_un(counter));
    P(ind) = counter;
end
end

function C = compute_partition(C_st,N) % Restore the partition from C_st
it = length(C_st);
C = zeros(1,N);
for counter_1 = 1 : N
    C(counter_1) = C_st{1,1}(counter_1);
    for counter_2 = 2 : it
        C(counter_1) = C_st{1,counter_2}(C(counter_1));
    end
end
end
```

B.5. Non-backtracking Method

```
function [Comm,N_cl,Q] = Non_back_tracking(A,N)
B_star = [A, eye(N) - diag(A*ones(N,1));eye(N) zeros(N)];       % Compute the 2N x 2N matrix
    B_star in eq. (27), page 14
Eigs = eigs(B_star,2*N);                                        % Compute the eigenvalues of
    B_star
ind = imag(Eigs) == 0;                                          % Identify real eigenvalues
N_cl = sum(Eigs(ind) > sqrt(max(Eigs)));                        % Determine the number of
    real eigenvalues larger that sqrt(lambda_1)
[U,~] = eigs(B_star,N_cl,'la');
U = real(U);
Comm = kmeans(U(1:N,:),N_cl);
Q = compute_modularity(A,Comm);
end

function Q = compute_modularity(A,C)
N = size(A,1);
Deg = A*ones(N,1);
Q = 1/nnz(A).*sum(sum((A - (1./nnz(A).*Deg*Deg')).*((ones(N,1)*abs(C)' - C*ones(1,N)) == 0)))
    ;
end
```

B.6. Modularity Eigengap

```
function [Comm,c,Q] = Eigengap(A)
N = size(A,1);
Deg = A*ones(N,1);
L_2 = sum(sum(A));
M = A - Deg*Deg'./L_2;
L = eigs(M,N);                              % Compute eigenvalues of Q
Eigs = sort(L,'ascend');                    % Sort the eigenvalues of Q
Eig_gap = flipud(diff((Eigs)));
[~,c] = max(Eig_gap(1:ceil(0.5*N)));        % Determine the largest gap
c = c + 1;
[U,~] = eigs(M,c,'la');
Comm = kmeans(U,c);
Q = compute_modularity(A,Comm);
end

function Q = compute_modularity(A,C)
N = size(A,1);
Deg = A*ones(N,1);
Q = 1/nnz(A).*sum(sum((A - (1./nnz(A).*Deg*Deg')).*((ones(N,1)*abs(C)' - C*ones(1,N)) == 0)))
    ;
end
```

B.7. Local Dominance

```
function [Comm,c,Q]=Local_Dominance(A)
G = graph(A);
D = degree(G);
N = size(A,1);
diG = [0,0];
l = zeros(1,N);

for i = 1:N
    Nb = neighbors(G,i); %find neighbors
    D_Nb = D(Nb); %degree of neighbors
    max_D_Nb = max(D_Nb); %max degree of neighbors
    index_D_Nb = []; %index of max
    for j = 1:length(D_Nb) %return the index, maybe more than 1
        if D_Nb(j)==max_D_Nb
            index_D_Nb = [index_D_Nb j];
        end
    end
    max_Nb = Nb(index_D_Nb); %neighbors with max degree
    for k = 1:length(max_Nb) %build digraph
        if D(max_Nb(k))>=D(i)&&max(ismember(diG,[max_Nb(k),i],'rows'))==0
            diG = [diG; [i, max_Nb(k)]];
            l(i)=1;
        end
    end
end
diG(1,:) = [];

%remove multiple out-going links
uni = unique(diG(:,1));
diG_u = [0,0];
for i = 1:length(uni)
    for j = 1:length(diG(:,1))
        if uni(i) == diG(j,1)
            diG_u(i,1) = uni(i);
            diG_u(i,2) = diG(j,2);
            break
        end
    end
end

C = setdiff(linspace(1,N,N),diG_u(:,1)); %local leaders
LL_D = D(C);
max_LL_D = max(LL_D);
```

```
index_LL_D = [];
for i = 1:length(LL_D)
    if LL_D(i)==max_LL_D
        index_LL_D = [index_LL_D i];
    end
end
M = C(index_LL_D); %local leaders with max degree

%links towards local leader with larger degree and shortest path
[~,index_sort] = sort(LL_D);
C_sort = C(index_sort);
for i = 1:length(C)-length(M)
    Distance = [];
    for j = i+1:length(C)
        [~,dis] = shortestpath(G,C_sort(i),C_sort(j));
        Distance = [Distance dis];
    end
    Dist_flip = fliplr(Distance);
    [min_Dis,ind_dis_flip] = min(Dist_flip);
    l(C_sort(i)) = min_Dis;
    ind_dis = length(Distance)-ind_dis_flip+1;
    diG_u = [diG_u; [C_sort(i), C_sort(ind_dis+i)]];
end

for i = 1:length(M)
    l(M(i)) = max(l);
end

D_star = D;
D_sorted = sort(D);
D_uni = unique(D_sorted);
for i = 1:length(D_star)
    for j = 1:length(D_uni)
        if D_star(i) == D_uni(j)
            D_star(i) = j;
        end
    end
end

l_star = l.^2;

for i = 1:length(C)
    delta(i) = ((l_star(C(i))-min(l_star))/(max(l_star)-min(l_star)))*((D_star(C(i))-min(
        D_star))/(max(D_star)-min(D_star)));
end

%assign community labels
Comm = [];
for i = 1:N
    if ismember(i,C)
        Comm(i) = i;
    else
        Comm(i) = diG_u(diG_u(:,1)==i,2);
    end
end

while(~isequal(sort(unique(Comm)),sort(C)))
    for i = 1:N
        if ~ismember(Comm(i),C)
            Comm(i) = diG_u(diG_u(:,1)==Comm(i),2);
        end
    end
end

uniComm = unique(Comm);
for i=1:length(uniComm)
    Comm(Comm==uniComm(i)) = max(uniComm) + i;
end
uniComm = unique(Comm);
for i=1:length(uniComm)
    Comm(Comm==uniComm(i)) = i;
```

```
114 end
115 c = max(Comm);
116 Q = compute_modularity(A,Comm');
117 end
118
119 function Q = compute_modularity(A,C)
120 N = size(A,1);
121 Deg = A*ones(N,1);
122 Q = 1/nnz(A).*sum(sum((A - (1./nnz(A).*Deg*Deg')).*((ones(N,1)*abs(C)' - C*ones(1,N)) == 0)))
    ;
123 end
```