CHALMERS
UNIVERSITY OF TECHNOLOGY

UNIVERSITY OF GOTHENBURG

Overlapped Community Detection in Multiplex Networks

Master's thesis in Computer science and engineering

Andreas Andersson

Department of Computer Science and Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
UNIVERSITY OF GOTHENBURG
Gothenburg, Sweden 2023

Master's thesis 2023

Overlapped Community Detection in Multiplex Networks

ANDREAS ANDERSSON

UNIVERSITY OF
GOTHENBURG

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Computer Science and Engineering
Chalmers University of Technology
University of Gothenburg
Gothenburg, Sweden 2023

Overlapped Community Detection in Multiplex Networks

ANDREAS ANDERSSON

Supervisor: Ashkan Panahi, Computer Science and Engineering
Advisor: Ashkan Panahi, Computer Science and Engineering
Examiner: Peter Damaschke, Computer Science and Engineering

Master's Thesis 2023
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Typeset in LaTeX
Gothenburg, Sweden 2023

Overlapped Community Detection in Multiplex Networks

ANDREAS ANDERSSON
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg

Abstract

Community detection is a fast-growing field in computer science, and it is easy to see why, as it enables the breakdown of complex networks into their principal components, which may accelerate the understanding of them. These intricate network structures frequently involve several relations and various kinds of component interactions. To fully leverage the dataset information, these complex systems are commonly represented as multiplex networks consisting of multiple layers to more explicitly model their multi-relational structure. In this thesis, we propose several extensions of two mono-layered community detection methods, namely belief propagation and cluster-driven low-rank matrix completion, in order to generalize them to the case of multiplex networks. These extensions include a flattening technique that converts the multiplex network into a mono-layered network by projecting the edges of each layer onto a single layer; a layer-by-layer technique that determines a consensus community structure by assembling the community structures obtained by running the algorithm for each layer; and lastly, a global extension that works directly on the multiplex network itself. The extended versions show an enhanced ability to detect communities in multiplex networks compared to their mono-layered equivalent; the increased detectability is even more prominent in sparse multiplex networks with a high number of overlaps.

Keywords: Computer, science, computer science, engineering, project, thesis.

Acknowledgements

I would like to thank my supervisor, Ashkan Panahi, for providing me with the opportunity to write this thesis and for giving me excellent guidance and advice, steering me in the right direction. To continue, I want to also thank Peter Damaschke for giving feedback regarding my thesis proposal and halftime report. Lastly, I want to show my gratitude to the department of Computer Science and Engineering at Chalmers for admitting me and providing me with an excellent education and for giving me the opportunity to write a Master's thesis in computer science.

Andreas Andersson, Gothenburg, 2023-06-12

Contents

List of Figures

List of Tables

1

Introduction

Today's society is becoming increasingly connected. Technical devices such as computers and mobile phones, in conjunction with the Internet, have enabled access to social networks that were previously out of reach. These modern social networks are often multi-relational and extremely broad, spanning countries all around the world and naturally comprising various forms of communities. Rarely are the communities completely isolated from one another, making it an increasingly difficult task to identify the underlying community structure. Community detection becomes even more complex when considering multi-relational networks consisting of several layers. These complexities highlight why community detection falls into the bucket of NP-hard problems, which can only be solved approximately by the use of approximation algorithms and heuristic methods [1].

1.1 Background

Community detection is one of the fundamental research topics in data science, and due to the growing number of available data networks, it is becoming increasingly relevant. Detecting an underlying community structure in a network, if it exists, is meaningful for several reasons. For example, it provides a large-scale image of the network, which enables analysis from a group-level perspective that is easier to interpret. The detection of critical components within complex networks has accelerated the number of advancements in several fields, including network analysis, locating interactive regions in brain networks, and identifying diseases that are highly similar to each other both genetically and symptomatically [2],[3],[4]. These social networks, neural circuits, and disease mappings share common attributes: they are often very complex and multi-relational, with several different types of interactions between components. To fully leverage the dataset information, these complex networks are commonly represented as multiplex networks consisting of multiple layers to more explicitly model their multi-relation structure. Unfortunately, existing community detection algorithms for traditional mono-layered networks are not sufficient to deal with the added complexities of multiplex networks. The three most common approaches for extending mono-layered community detection methods to the case of multiplex networks include layer aggregation, ensemble methods, and global methods that operate directly on the multiplex network. Community detection in multiplex networks is a relatively new and unresearched subject, and many mono-layered community detection techniques remain to be extended to mul-

tiplex networks. In this project, the aim is to extend existing community detection algorithms to multiplex networks using several different extension techniques, such as flattening, ensemble methods, and global extensions [5].

1.2 Problem

The main purpose of this research is to generalize two existing mono-layered community detection methods, namely belief propagation and cluster-driven low-rank matrix completion, and examine their performance on both synthetic data and some real-world word networks. We chose to extend these two methods in particular for two reasons. Firstly, these methods have proven successful for mono-layered community detection problems in [6] and [7], and may therefore be equally effective for detecting multiplex community structures. Secondly, these are two vastly different approaches for detecting community structures, which will hopefully highlight the complexities behind the problem of community detection. A global approach to belief propagation has been implemented for multiplex networks by [8]. The properties of this extension are further examined in this project and compared to other extension techniques. The performance of community detection methods may depend on several different factors, including the sparsity of the network, the average node degree, the number of communities, the number of overlaps, the number of layers, and the community sizes [9],[10]. To examine the limitations of belief propagation and low-rank matrix completion, they are tried and tested on synthetic data where each of the above-mentioned properties is isolated.

1.3 Thesis structure

The thesis is structured in the following way: First, in Section 2.1 the problem of community detection is defined for simple mono-layered networks. To continue, we introduce the mono-layered versions of belief propagation and low-rank matrix completion in Section 2.2. Section 2.3.1 describes the multiplex network data structure in more detail and Section 2.3.3 extends the problem of community detection to the case of multiplex networks. Section 2.3.5 introduces different extension techniques for generalizing community detection methods to multiplex networks. Chapter 3 describes the methodology used in this project and Chapter 4 presents the obtained results for both synthetic and real-world data. Lastly, Chapter 5 gives a discussion and conclusion to the project by answering the problem formulation.

2

Theory

2.1 Community detection in mono-layered networks

The aim of community detection is to partition a given network into densely intraconnected components with few overlaps. These networks are most naturally represented as graphs. A graph is a pair, $G = (V, E)$, where V is a set of nodes, $V = \{v_1, ..., v_n\}$ and E is a set of edges that signify connections between nodes. The goal of community detection can be described as follows: Given a graph, $G = (V, E)$, community detection aims to determine a partition of the nodes $C = \{C_1, ..., C_k\}$ consisting of k communities that are densely intraconnected with few overlapping edges between distinct communities. The simplest case of community detection is when there are no overlapping edges between communities. In this case, community detection is equivalent to finding the connected components in the network, which can be done in linear time by repeated use of breadth-first search. However, this approach fails completely when there are overlapping edges between communities, which is a common scenario in practice. Hence, the approach relying on BFS is discarded and we instead opt for approximation algorithms and heuristic methods to solve the NP-hard problem of community detection [11].

2.2 Community detection algorithms

Here we describe the approximation algorithms belief propagation (BP) and cluster-driven low-rank matrix completion (CLMC) for detecting community structures in mono-layered networks.

2.2.1 Belief propagation

In this section, we introduce the belief propagation algorithm (BP) for detecting communities in mono-layered networks. Since BP is an algorithm for estimating marginal probabilities, we first formulate the community detection problem in terms of a Bayesian inference problem. All successful Bayesian inference approaches require some assumption about the underlying probability distribution. This assumption is described in detail in Section 2.2.1.1. The second step of BP consists of translating the given network into a so-called factor graph, a network structure that is described in Section 2.2.1.2. Finally, in Section 2.2.1.3 the BP algorithm is

presented.

2.2.1.1 The stochastic block model

Nodes that share common attributes tend to gravitate toward one another, whereas dissimilar nodes repel. Naturally, the network divides into densely connected communities of similar nodes, with only a few overlaps between different communities. This observation, while unlikely to capture the full intricacy of community formation, can serve as a template for developing an effective generative model for generating networks with well-defined community structures. The main goal of the generative models is not (surprisingly) to generate networks but rather to serve as a fitting function for detecting the underlying community structure within a given network [12]. In this paper, we study the stochastic block model (SBM), the most widely used generative model for community detection.

The stochastic block model SBM(C, T) is defined by a set of vertices $V = \{v_1, ..., v_n\}$, set of community labels $\{1, ..., k\}$, *membership matrix* $T = (\mathbf{T}_{v_1}, ..., \mathbf{T}_{v_n})^\intercal$ and a *block matrix* C. Each vertex $v \in V$ is said to belong to one of k communities and its community label is stored in its corresponding membership vector $T_v = (t_1(v), ..., t_k(v))$ where

$$t_i(v) = \begin{cases} 1, & \text{if } v \text{ belong to community } i, \\ 0, & \text{otherwise,} \end{cases} \qquad 1 \leq i \leq k. \tag{2.1}$$

The block matrix, C, is a $k \times k$-matrix with elements $C_{i,j} \in [0, 1]$, $1 \leq i, j \leq k$ corresponding to the probability that any two vertices $v \in V$ and $u \in V$ that belong to community i and j respectively, are connected.

Given this definition, the stochastic block model is able to generate community structures exhibiting a wide array of different properties, including varying community sizes, different levels of sparsity, and the number of overlaps between communities [13].

The community sizes, N_i, $1 \leq i \leq k$, can be determined by taking the sum of the elements in column i of T. To continue, the number of edges between groups can be derived from A and T through a series of matrix multiplications: $E' = T^\intercal AT$. Note that the number of interconnections within groups is counted twice. To account for this, we introduce an $K \times K$-*group connectivity matrix*, E, with the following elements:

$$E_{i,j} = \begin{cases} E'_{i,j}, & \text{if } i \neq j, \\ \frac{1}{2}E'_{i,j}, & \text{otherwise.} \end{cases} \tag{2.2}$$

The number of possible distinct edges within a group i is $\binom{N_i}{2}$. Thus, the edge density within group i is given by

$$\frac{E_{i,i}}{\binom{N_i}{2}}. \tag{2.3}$$

Edge densities between two different groups, i and j, are calculated slightly differently as the number of possible distinct edges is instead $N_i \cdot N_j$. As a result, the edge density between distinct groups i and j is instead given by

$$\frac{E_{i,j}}{N_i \cdot N_j}. \tag{2.4}$$

Let $\mathbf{D}$ denote the $K \times K$-*edge density matrix* with the following elements

$$D_{i,j} = \begin{cases} \frac{E_{i,j}}{\binom{N_i}{2}}, & \text{if } i = j, \\ \\ \frac{E_{i,j}}{N_i \cdot N_j}, & \text{otherwise.} \end{cases} \tag{2.5}$$

Thus, the density matrix $\mathbf{D}$ describe the proportion of vertices that are connected within communities as well as the degree of connectivity across various communities [14].

As the SBM is mostly a generative model, we include an illustrative example that displays the inherent properties of the SBM-generated graph. Furthermore, the concrete example will serve as a bridge between the SBM and the subject of community detection to show how they relate to each other.

In Figure 2.1, we see the adjacency matrix of a graph generated by the SBM(C, T) with the following parameters: $k = 3$,

$$\begin{gathered} t_1(v_p) = \begin{cases} 1, & \text{for } 1 \leq p \leq 20, \\ 0, & \text{otherwise} \end{cases}, \quad t_2(v_p) = \begin{cases} 1, & \text{for } 21 \leq p \leq 50, \\ 0, & \text{otherwise} \end{cases}, \\ t_3(v_p) = \begin{cases} 1, & \text{for } 51 \leq p \leq 100, \\ 0, & \text{otherwise} \end{cases}, \quad C = \begin{pmatrix} 0.95 & 0.05 & 0.05 \\ 0.05 & 0.95 & 0.05 \\ 0.05 & 0.05 & 0.95 \end{pmatrix}. \end{gathered} \tag{2.6}$$

The most apparent property of the plotted adjacency graph is its block-like structure, hence the name stochastic **block** model. This structure is partly a consequence of our choice of parameters (2.6). By conveniently labeling the nodes in the same arrangement as their node numbers, they are plotted next to their community members. Furthermore, as the diagonal elements of C in (2.6) are significantly larger than their non-diagonal elements, communities are densely intraconnected, while the overlaps between distinct communities are few. Hence, the block-like structure emerges in the plotted adjacency matrix in Figure 2.1.

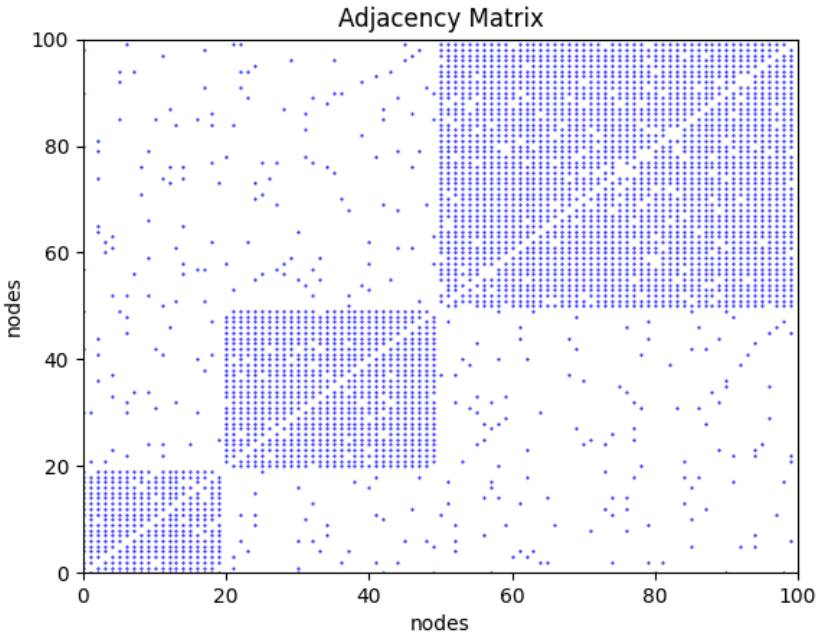


Figure 2.1: Adjacency matrix of the SBM-generated graph with parameters in (2.1).

In practice, the node numbers and their community labels are rarely aligned, resulting in a considerably more chaotic adjacency matrix plot where the communities are difficult to differentiate. In Figure 2.2, we see the identical adjacency matrix as in Figure 2.1, except the node numbers have been shuffled randomly. As the Figures 2.1 and 2.2 display the same graph with different node numbers, it is apparent that the node numbers can be rearranged such that the adjacency matrix in Figure 2.2 becomes the same block-model as in Figure 2.1. This is the point at which the two ideasthe SBM and community detectionintertwine. The block-like structure seems to be an inherent property of any network containing a community structure [15]. Naturally, we adopt a Bayesian inference approach for community detection under the assumption that the observed network is generated from an underlying stochastic block model, SMB(C,T), with some parameters C and T. In light of this presumption, the goal is to find the maximum a-posteriori (MAP) estimate of the community structure C and its corresponding community labels $T = \{t_i\}$ given an observed network W and a known SBM with parameters θ from which it was generated, that is,

$$\hat{T}_{MAP} = \arg\max_{T} P(T|W, \theta). \tag{2.7}$$

By applying Bayes rule to 2.7 we have

$$\hat{T}_{MAP} = \arg\max_{T} P(T|W, \theta) = \frac{P(W, T|\theta)}{\sum_{\{t_i\}} P(W, \{t_i\}|\theta)}. \tag{2.8}$$

In the context of Bayesian inference, $P(T|W, \theta)$ is referred to as the a-posteriori distribution, and $P(W, \{t_i\}|\theta)$ is called the likelihood function. Intuitively, the likelihood function, $P(W, \{t_i\}|\theta)$ is the probability that a stochastic block model with

parameters $\theta = \{C, k, \{N_i\}_{i=1}^k\}$ generates an observed network with community labels $\{t_i\}$. Generating a network from a given SBM can be thought of as a series of events, one for each pair of nodes, with two outcomes: either an edge is generated between them or not. The probability of generating an edge connecting a given pair of nodes (i, j), $i \in V$, $j \in V$ is given by c_{t_i,t_j}. Hence, the likelihood function for generating a network W with community labels T conditioned on the SBM-parameters θ is given by

$$P(W, T|\theta) = \frac{1}{Z} \prod_{1 \leq i < j \leq n} (T_p^\intercal C T_q)^{A_{ij}} \cdot (1 - T_p^\intercal C T_q)^{1-A_{ij}} \cdot \prod_i n_{t_i}. \tag{2.9}$$

Unfortunately, it is often too computationally demanding to calculate $\hat{T}_{MAP}$, in practice. In order to ease computations, we therefore elect to calculate the marginal maximum a-posteriori probability (MMAP) estimate for each node individually instead of directly calculating $\hat{T}_{MAP}$ [6]. The marginal probability distribution for the node v_i is given by

$$p_i(\tau) = P(t_i = \tau|W, \theta) = \sum_{\{t_j\}:t_i=\tau} P(T|W, \theta). \tag{2.10}$$

Hence, the MAAP estimate for node i is given by

$$\hat{t}_{MAAP} = \arg\max_\tau p_i(\tau). \tag{2.11}$$

We denote the collection of MAAP estimates for each node in the network by T_{MAAP}, that is

$$\hat{T}_{MAAP} = \{\hat{t}_{i,MAAP}\}. \tag{2.12}$$

2.2.1.2 Factor graphs

At first glance, the problem of community detection can seem daunting, as the objective function typically involves a large number of variables. However, due to local properties, complex problems such as community detection can often be broken down into smaller pieces, which (hopefully) by themselves are easier to solve. In this section, we define belief propagation - a local message-passing algorithm that utilize these local properties to solve the problem of community detection. Before doing so, we introduce some necessary components for composing the algorithm. Variants of the belief propagation algorithm exist for a variety of graphical models, including Bayesian networks and Markov random fields [16]. While these versions have proven useful in many applications, we choose to direct our focus on the version operating on factor graphs as the community detection problem can naturally be represented in terms of factor graphs [8].

A factor graph represents the factorization of a function. In particular, it can be utilized to represent probability distribution functions, allowing for efficient computation of marginal probability distributions through the use of belief propagation [16]. A factor graph consists of edges and two types of nodes: function nodes (also called local functions) and variable nodes. To distinguish between them, factor

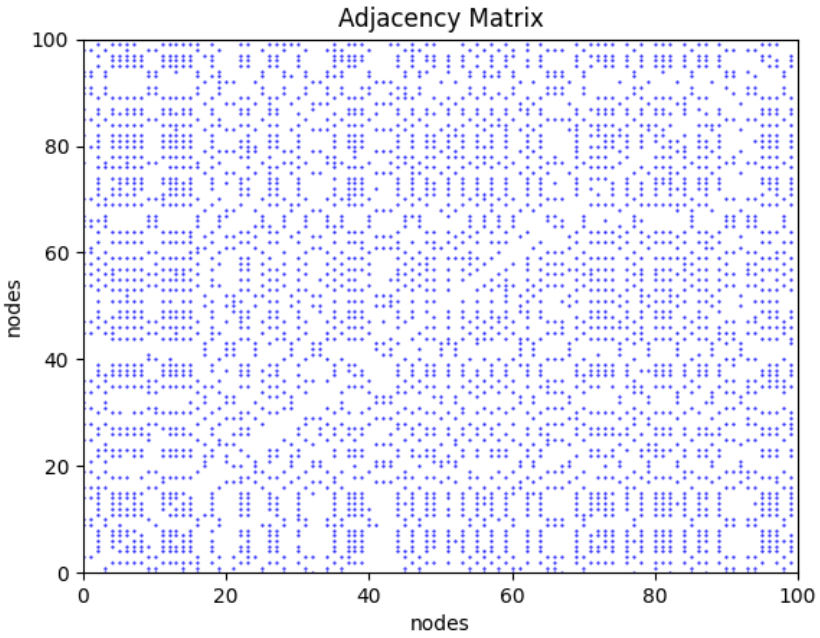


Figure 2.2: Adjacency matrix of the SBM-generated graph with parameters in (2.1) with a shuffled node ordering.

nodes and variable nodes are represented by squares and circles, respectively (see Figure 2.3). Edges only exist between factor nodes and variable nodes. If a given local function, g is connected to a given variable node, x it means that g is a function of x. When modeling a network with a community structure, we only consider the local functions of two variables, as a network can be described in terms of pairwise interactions between nodes through edges. To continue, when we refer to local functions, we refer to local functions of two variables. Suppose that a given global function $f(v, w, x, y, z)$ can be factored as the following expression of local functions:

$$f(v, w, x, y, z) = f_1(v, w)f_2(w, x)f_3(x, y)f_4(y, z). \tag{2.13}$$

Then the factorization can be expressed by the factor graph in Figure 2.3.

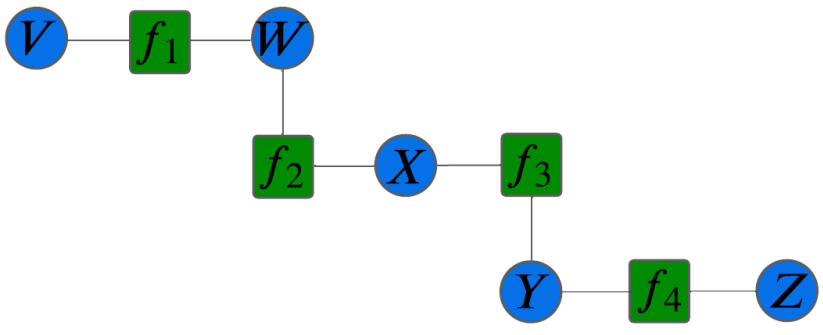


Figure 2.3: The factor graph representation of 2.13.

The depiction in 2.3 clearly shows how a joint probability distribution can be represented graphically as a factor graph. However, the most vital part still remains, namely how to utilize this representation in order to compute the marginal probabilities. To answer this question, we give a brief review of marginal probability distributions. Given a joint probability distribution of N random variables $X_1, X_2, ..., X_N$ the marginal distribution of X_1 is the probability distribution of X_1 when the values of $X_2, ..., X_N$ are not taken into account - we say that $X_2, ..., X_N$ is marginalized. More precisely, the marginal probability distribution of X_1 is given by

$$p_X(x_{i_1}) = \sum_{i_2} ... \sum_{i_N} p(x_{i_1}, y_{i_2}, ..., y_{i_N}). \tag{2.14}$$

Given the definition in (2.14), to calculate the marginal probabilities for X, we have to sum over all configurations of variable Y. A *configuration of variables* is a particular assignment of variable values. To illustrate, let $X = \{x_i\}_{i \in N}$ be a collection of variables, where each variable x_i takes on a value from the state space $\mathcal{X}_i$ of x_i. Then the particular assignment $\omega = \{x_1 = a_1 \in \mathcal{X}_1, ..., x_N = a_N \in \mathcal{X}_N\}$ is a configuration of variables. A configuration of variables, ω, is said to be **valid** for a global function f if and only if $f(\omega) \neq 0$. The configuration space for a global function f is the set of all valid configurations and is denoted by Ω. In other words, to compute the marginal distribution for a given variable, we have to consider all valid configurations for the marginalized variables. Unfortunately, calculating marginal probabilities is an NP-hard problem as the number of configurations grows exponentially with regards to the number of marginalized variables, making it inefficient to compute even for a small number of variables [17]. This is where belief propagation comes into play as an efficient way of approximately calculating the marginal probabilities.

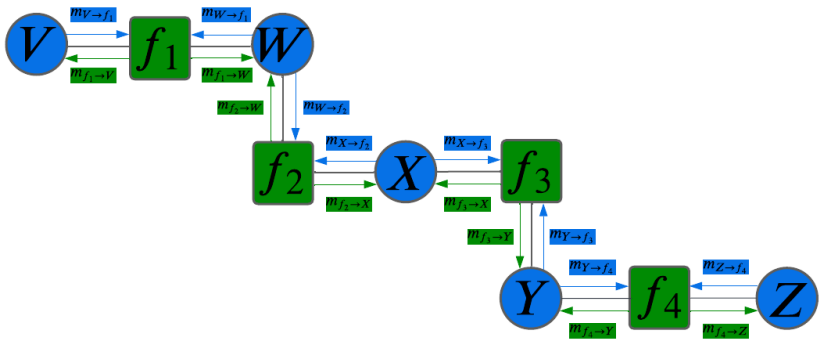


Figure 2.4: A visualization of the messages for the factor graph representation of 2.13.

As previously mentioned, belief propagation is a local message-passing algorithm on graphical models, passing messages between adjacent nodes in the graph. In the case of factor graphs, there are two types of messages that need to be distinguished: those that go from variable nodes to factor nodes and those that go from factor nodes to variable nodes, see Fig. 2.4. The message from a variable node x_i to a

factor node f_a is updated recursively by calculating the product of all incoming messages to x_i from all adjacent factor nodes except for the reciever of the message, f_a, that is

$$m_{i\to a}(x_i) = \frac{1}{Z_{i\to a}} \prod_{f_c \in N(x_i)\setminus f_a} m_{c\to i}(x_i). \tag{2.15}$$

Here $N(x_i)$ denotes the neighboring factor nodes for x_i and $Z_{i\to a}$ is a normalization factor so that $\sum_{x_i\in\mathcal{X}} m_{i\to a}(x_i) = 1$. As messages are non-negative and sum to one they can be interpreted as probabilities. Similarly, the message from a factor node f_a to a variable node x_i is defined as the product of all incoming messages to f_a from all adjacent variable nodes except from x_i itself marginalized over all associated variables except x_i, that is,

$$m_{a\to i}(x_i) = \sum_{\mathbf{x}\setminus x_i} f_j(\mathbf{x}_j) \prod_{x_k \in N(f_a)\setminus x_i} m^{i\to a}(x_i). \tag{2.16}$$

To illustrate how the message-passing process of BP operates, we consider the factor graph in 2.3 and calculate the marginal probability for Y. By definition, the marginal probability of Y is given by

$$P(Y = y) \propto \sum_{x,v,w} f(V = v, W = w, X = x, y, Z = z). \tag{2.17}$$

By utilizing the underlying structure of the graph in Figure 2.3 we have

$$P(y) \propto \sum_{v,w,x,y} f_1(v, w) \cdot f_2(w, x) \cdot f_3(x, y) \cdot f_4(y). \tag{2.18}$$

The key trick of belief propagation is to push the summations through the variables they do not sum over, allowing for a much more efficient way of computing the marginalization than summing over all the marginalized variables. We will refer to this technique as "summation pushing". In the case of our example, BP calculates the marginalized function (also called belief) as follows:

$$\begin{aligned}
P(y) &\propto \sum_{v,w,x,z} f_1(v, w) \cdot f_2(w, x) \cdot f_3(x, y) \cdot f_4(y, z) \\
&= \sum_{w,x,z} f_2(w, x) \cdot f_3(x, y) \cdot f_4 \left[\sum_v f_1(v, w)\right] \\
&= \sum_{w,x,z} f_2(w, x) \cdot f_3(x, y) \cdot f_4 m_{f_1\to w}(w) \\
&= \sum_{w,x} f_2(w, x) \cdot f_3(x, y) \cdot m_{f_1\to w}(w) \left[\sum_z f_4(y, z)\right] \\
&= \sum_{w,x} f_2(w, x) \cdot f_3(x, y) \cdot m_{f_1\to w}(w) \cdot_{f_4\to y} (y) \\
&= \sum_x f_3(x, y) \cdot_{f_4\to y} (y) \cdot \left[\sum_w f_2(w, x) \cdot m_{f_1\to w}(w)\right] \\
&= \sum_x f_3(x, y) \cdot_{f_4\to y} (y) \cdot m_{f_2\to x}(x) \\
&= m_{f_4\to y}(y) \cdot \left[\sum_x f_3(x, y) \cdot m_{f_2\to x}(x)\right] \\
&= m_{f_4\to y}(y) \cdot m_{f_3\to y}(y).
\end{aligned}$$

As shown by the above calculation, after convergence of BP, the belief can be approximately computed by multiplying all the incoming messages; thus the belief of node i is given by

$$b_i(x_i) \propto \prod_{f_a \in N(x_i)} m_{a \to x_i}(x_i). \tag{2.19}$$

The summation pushing process reduces the time complexity significantly compared to directly summing over all marginalized variables. Consider a factor graph of consisting of k variables $x_1, ..., x_k$ with state spaces $x_1, ..., x_k \in \{1, ..., N\}$. Without summation pushing, computing a marginalized function requires one to compute the probabilities of all N^k configurations, taking a total of $O(N^k)$ time. However, when utilizing summation pushing, the computation consist of $k - 1$ summations over one variable. For each summation a total of N^2 terms is summed up as only local functions of two variables are considered, taking a total of $O(k \cdot N^2)$ time.

Belief propagation works remarkably well for tree-structured factor graphs, that is, graphs without loops. In fact, it is proven that BP provides an exact solution for tree graphs [16]. However, when loops are present, the algorithm is no longer exact and may require infinite updates to converge. This becomes apparent when considering the recursive update formulas in (2.15) and (2.16). Instead of abandoning belief propagation entirely, the message updating procedure is adjusted slightly in hopes that BP converges to a fixed-point solution. Indeed, by updating messages in a random order, BP has shown to produce good results even for loopy graphs. However, even when heuristically updating messages, BP occasionally fail to converge for some loopy graphs and instead oscillates between several states. The conditions for which BP fails to converge are yet not well-understood [18].

2.2.1.3 The BP algorithm for community Detection in mono-layered networks

To be able to apply BP, it is required that the proposed model satisfy two criteria. Firstly, there must exist a natural correspondence between some given probability function $P(x)$ and the given graphical model. Secondly, the constructed factor graph needs to only consider pairwise constraints. The groundwork for criteria (1) was established in Section 2.2.1.1 where we show that the problem of community detection can be formulated in terms of the marginal probabilities (2.10). In order to fulfill criteria (2), we construct a factor graph consisting of variable nodes $x_1, ..., x_N$, equivalent to the nodes in the observed network. The state space for each variable node x_i is given by $\mathcal{X}_i = \{t_1, ..., t_N\}$, that is, the possible community labels in our observed network. The factor nodes in the factor graph correspond to the edges and non-edges in the network

$$f(x_i, x_j) = \begin{cases} c_{t_i,t_j} & \text{if} \quad (i,j) \in E, \\ 1 - c_{t_i,t_j} & \text{if} \quad (i,j) \notin E. \end{cases} \tag{2.20}$$

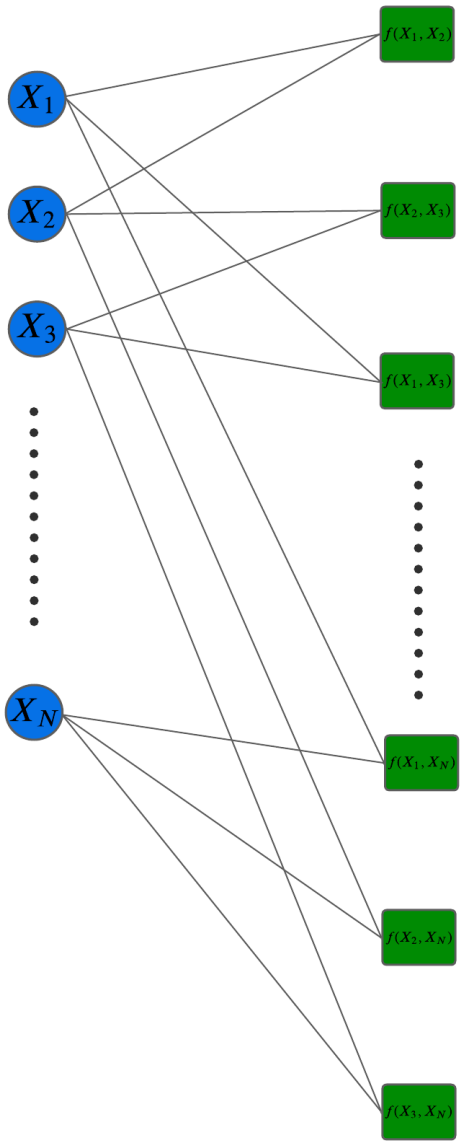


Figure 2.5: The layout of the factor graph utilized in the BP algorithm.

An advantageous property of the constructed factor graph in Figure 2.5 is that each factor node has exactly two adjacent variable nodes. Hence, we can define a message function that passes messages directly between variable nodes. To realize this, we examine what happens to a message that is sent from a variable node x_i to another variable node x_j. A message traveling from x_i to x_j must go along the path that goes through the factor node node $f(x_i, x_j)$. However, as this factor node has no adjacent variable nodes other than x_i and x_j there are no other nodes that may disturb the message passing process. The question is now, what happens to the message at the intermediary factor node before it is passed to the variable node x_j. According to the update functions (2.16) the incoming message from the transmitter is multiplied by the corresponding local function $f(x_i, x_j)$ before traveling to the reciever. Thus,

we get the following update rule for direct messages between variable nodes:

$$m_{i\to j}^{t_i} = \frac{1}{Z_{i\to j}} n_{t_i} \prod_{k\in N(i)\setminus j} \left[\sum_{t_k} c_{t_k t_i}^{A_{ik}} (1 - c_{t_k t_i})^{1-A_{ik}} m_{k\to i}^{t_k} \right]. \tag{2.21}$$

Since each pair of variable nodes interacts, either via a factor node corresponding to an edge or a non-edge, the computation in (2.21) is relatively expensive, taking $O(N^2)$ time. In [6] the time complexity is reduced by assuming that each variable node x_i sends the same message to all its non-neighboring variable nodes x_j by introducing an external field h. The idea is that two nodes that are not connected have little knowledge about the state of the other, and thus have little influence in determining its community label, making the approximation error small. We get the following update rule:

$$m_{i\to j}^{t_i} = \frac{1}{Z_{i\to j}} n_{t_i} e^{-h_{t_i}} \prod_{k\in N(i)\setminus j} \left[\sum_{t_k} c_{t_k t_i} m_{k\to i}^{t_k} \right], \tag{2.22}$$

where the external field is given by

$$h_{t_i} = \frac{1}{N} \sum_k \sum_{t_k} c_{t_k t_i} b_{t_k}^k. \tag{2.23}$$

Here $b_{t_i}^k$ denotes the belief at node x_k for the community label t_k given by

$$b_i^{t_i} = \frac{1}{Z_i} n_{t_k} e^{-h_{t_k}} \prod_{k \ inN(i)} \sum_{t_k} c_{t_k t_i} m_{k\to i}^{t_k}. \tag{2.24}$$

Lastly, the community labels for each nodes is determined by the largest component of its belief, that is

$$t_i = \arg\max_q b_i^q. \tag{2.25}$$

By assembling all the pieces, we get the BP algorithm in Algorithm 1 for community detection in accordance with [6].

Algorithm 1 Belief Propagation for Community Detection in Single Layered Networks

Input: Adjacency matrix A, block matrix C, community sizes n_i, number of communities q, convergence parameter ϵ, maximum number of iterations $t_{\max}$
Output: Community labels
Initialize: Random ormalized beliefs b_i, external field h, messages $m_{i\to j}$, $i = 1, ..., n$, $j \in N(i)$
while not conv $= |\sum m_{\text{new}} - m_{\text{old}}| < \epsilon$ and $t < t_{\max}$ **do**
  for every layer l **do**
    Pick a random ordering, O_m, of messages
    for every message m (in order O_m) **do**
      if if m is intra-layered message **then**
        Update $m = m_{i\to j}$ according to
      end if
    end for
    for every message $m_{i\to j}$ (in order O_m) **do**
      Update the $q-$components of $m_{i\to j}$ according to (2.22)
      Update all $q-$components of b_j according to (2.24)
      Update external field h according to (2.23)
    end for

    Return: Community labels $q_i = \arg\max_q b_i^q$, $i = 1, ..., n$

2.2.2 Cluster-driven Low-rank Matrix Completion

In this section, a low-rank matrix completion algorithm is described and how it relates to community detection. Low-rank matrix completion is a vastly different community detection approach compared to belief propagation. Instead of estimating the marginal probabilities under the assumption that the network originates from an underlying stochastic block model, the problem of community detection is formulated as a convex optimization problem. The final objective function is derived in several steps, starting from the matrix rank minimization problem (RMP) which is described in Section 2.2.2.1. It turns out that the clique problem is closely related to community detection; thus, in Section 2.2.2.2 the clique problem (CP) is expressed in terms of the RMP in accordance with [19]. Since the clique problem is a strict version of community detection that only works on fully connected block-diagonal matrices, we introduce the cluster-driven low-rank matrix completion (CLMC) algorithm in Section 2.2.2.3 which aims to construct such a matrix. Lastly, in Section 2.2.3 the impact of the choice of parameters is described.

2.2.2.1 The rank minimization problem

Before introducing the *matrix rank minimization problem* we give brief description of *Occam's razor* as it is the foundational principle upon which RMP is built. Occam's razor, also known as the *law of parsimony*, serves as a general principle by which we select among candidate theories with similar or identical performance. It states that

among competing explanations, the simpler one, is to be favored. Occam's razor is applicable to a wide range of problems, including model selection, where the goal is to find the model of lowest dimensionality that fits the data well [20].

Dimensionality is often analogous to the complexity and solving time of a given problem. Therefore, in the essence of Occam's razor, given a set of candidate models, it is generally desirable to pick the model with the smallest dimensionality. If the set of proposed models satisfy a set of convex constraints, then picking the simplest model can be expressed as a RMP [21]. The goal of matrix rank minimization is to find a matrix A of lowest dimensionality satisfying a set of constraints $\mathcal{C}$, giving us the following optimization problem:

$$\begin{aligned} \min_{A} \quad & \mathbf{rank}(A) \\ \text{s.t.} \quad & A \in \mathcal{C}. \end{aligned} \tag{2.26}$$

Unfortunately, RMP falls into the bucket of NP-hard problems. Hence, in order to solve RMP in an acceptable amount of time, we have to rely on heuristic or approximation methods. In [22] a convex relaxation technique is considered in order to efficiently solve (2.26) approximately. Convex relaxation is a heuristic optimization technique that aims to approximate the original (hard) problem such that it becomes convex. In the case of the RMP, the rank function can be relaxed using the *nuclear norm*, which is currently the best known approximation over the unit ball of matrices with a norm less than one [19]. The nuclear norm is defined as

$$||A||_* = \sum_{i=1}^{\min(m,n)} \sigma_i(A), \tag{2.27}$$

where A is an $m \times n$-matrix and $\sigma_i(A)$ is its singular values. A nice property of the nuclear norm is that it is a convex function and can thus be efficiently optimized.

2.2.2.2 Expressing CP in terms of RMP

We are now familiar with the RMP. However, the main question remains: How is RMP related to community detection? It turns out that RMP and the clique problem (CP) are closely related and that there exists a natural translation from CP to RMP [19]. The CP is essentially a strict version of community detection, as the goal is to find fully connected communities where each member is connected to each other. These fully connected communities are seldom found in real networks, but they can serve as a starting point for developing an efficient community detection algorithm.

To be able to express CP in terms of RMP, we first make a slight adjustment to the traditional adjacency matrix. Let A denote the adjacency matrix of a simple graph $G = (V, E)$. Traditionally, if $A_{i,j} = 1$, for $i = j$, then v_i is said to be connected to itself by a loop. As we only examine simple graphs, that is, graphs without any loops, we have that $A_{i,j} = 0$ for all i, j where $i = j$. Therefore, for any clique K of

size n in G, the adjacency matrix, A^K, of the induced subgraph $G[K]$ satisfy

$$A^K_{i,j} = \begin{cases} 1, & \text{if } i \neq j, \\ 0, & \text{otherwise.} \end{cases} \tag{2.28}$$

This definition presents a problem, namely that the row vectors of $\hat{A}$ are linearly independent. Therefore, the rank of $\hat{A}$ is equal to n. As G is a simple graph, we can make a slight adjustment to the adjacency matrix that retains the one-to-one relationship between the adjacency graph and its corresponding graph representation. Let $\hat{A}$ denote the adjusted adjacency matrix defined as follows:

$$\hat{A}_{i,j} = \begin{cases} 1, & \text{if } v_i \text{ and } v_j \text{ are connected, or } i = j, \\ 0, & \text{otherwise.} \end{cases} \tag{2.29}$$

For the induced subgraph $G[K]$, we have that the adjusted adjacency matrix $\hat{A}^K$ is of rank one as all of its elements are equal to one, making all of its rows linearly dependent.

A clique is a submatrix of rank one. Hence, a clique of size n can be found by solving the following RMP:

$$\begin{aligned} \min_{\hat{A}} \quad & \mathbf{rank}(\hat{A}) \\ \text{s.t.} \quad & \sum_{i \in V} \sum_{j \in V} \hat{A}_{i,j} \geq n^2, \\ & \hat{A}_{i,j} = 0 \text{ if } (i,j) \notin E \text{ and } i \neq j, \\ & \hat{A}_{i,j} \in [0,1]. \end{aligned} \tag{2.30}$$

which can be expressed in its relaxed form using the nuclear norm as an approximation of the rank function:

$$\begin{aligned} \min_{\hat{A}} \quad & ||\hat{A}||_* \\ \text{s.t.} \quad & \sum_{i \in V} \sum_{j \in V} \hat{A}_{i,j} \geq n^2, \\ & \hat{A}_{i,j} = 0 \text{ if } (i,j) \notin E \text{ and } i \neq j, \\ & \hat{A}_{i,j} \in [0,1]. \end{aligned} \tag{2.31}$$

2.2.2.3 The cluster-driven low-rank matrix completion algorithm

As previously mentioned, the CP is closely related to the problem of community detection, as a clique is a community where each member is connected to each other. However, it turns out that CP is too strict of a definition to detect communities, as communities in practice are rarely fully connected. In this section, in accordance with [7] we define an algorithm that aims to repair the given network, such that it consists of perfect communities by adding the missing edges within each community and removing the overlaps between communities.

More precisely, starting with an initial network W consisting of n nodes, the goal is to gradually decompose it into three components, $\hat{C}$, R and E such that $A = \hat{C}+R+E$. The first component, $\hat{C}$, is an $n \times n-$clustering matrix that, after decomposition, will be similar to the initial network with one key difference: It does not include the overlapping edges. The overlapping edges are instead stored in the second component, R, which is an $n \times n-$noise matrix that only contain overlapping edges. The final component, E, is a $n \times n$-supplement matrix that includes the missing intraconnections within communities.

In order to decompose an initial network W of size n with an adjacency matrix A, we formulate a convex optimization problem with the appropriate constraints that enforce $\hat{C}$,R and E to take the above-mentioned shapes. Ideally, after decomposition, the clustering matrix should consist of k cliques, where k is the number of communities. In other words, the clustering matrix is expected to be a block matrix consisting of k blocks. As the connectivity within communities is assumed to be dense, it is safe to assume that the clustering matrix is low rank. This gives us the following optimization problem:

$$\begin{aligned} \min \quad & \text{rank}(\hat{C}) \\ \text{s.t.} \quad & A = \hat{C} + R + E, \\ & \hat{C} \geq 0, \quad R \geq 0. \end{aligned} \tag{2.32}$$

The sole purpose of the supplement matrix E is to complement $\hat{C}$ with its missing edges. To avoid that E alters any existing connections within the network, we introduce a constraint that enforce that the elements, $E_{i,j}$, of E are zero if the edge exists in the original network, that is, if $A_{i,j} = 1$. This constraint can be formulated in terms of the linear operator $\pi_\Omega : \mathbb{R}^{n\times n} \to \mathbb{R}^{n\times n}$ that leaves the entries of Ω unaltered and sets those outside to zero. By introducing the constraint, we get

$$\begin{aligned} \min \quad & \text{rank}(\hat{C}) \\ \text{s.t.} \quad & A = \hat{C} + R + E, \quad \pi_\Omega(E) = 0, \\ & \hat{C} \geq 0, \quad R \geq 0, \end{aligned} \tag{2.33}$$

where $\Omega = \{(i, j) : A_{i,j} = 1\}$.

As shown in Section 2.2.2.1, we can approximate the rank function using the nuclear norm, yielding the following optimization problem

$$\begin{aligned} \min \quad & ||\hat{C}||_* \\ \text{s.t.} \quad & A = \hat{C} + R + E, \quad \pi_\Omega(E) = 0, \\ & \hat{C} \geq 0, \quad R \geq 0. \end{aligned} \tag{2.34}$$

The optimization problem in (2.34) steers the decomposition process in the right direction, making sure $\hat{C}$ becomes a block-diagonal matrix consisting of low rank

blocks. However, (2.34) does not guarantee that the number of blocks in $\hat{C}$ corresponds to the number of communities in the network. In order to impose an appropriate constraint that ensures this property, the *Laplacian matrix* is introduced. The Laplacian matrix is defined as $L_{\hat{C}} = D_{\hat{C}} - \hat{C}$, where $D_{\hat{C}}$ is the degree matrix of $\hat{C}$, that is

$$D_{\hat{C}} = \begin{pmatrix} \deg(v_1) & 0 & \dots & 0 \\ 0 & \deg(v_1) & & \\ \vdots & & \ddots & \\ 0 & & & \deg(v_n) \end{pmatrix}. \tag{2.35}$$

Thus the Laplacian matrix of $\hat{C}$ is given by

$$L_{\hat{C}}(u,v) = \begin{cases} \deg(u), & \text{if } u = v, \\ -\hat{C}(u,v), & \text{if } u \text{ and } v \text{ are adjacent,} \\ 0, & \text{otherwise.} \end{cases} \tag{2.36}$$

The Laplacian matrix was introduced in order to make sure that the clustering matrix will consist of k blocks; this is accomplished by the following theorem.

Theorem 1: If $\hat{C}$ is non-negative, the rank of the Laplacian matrix $L_{\hat{C}}$ is equal to $n - k$, where n is the size of $\hat{C}$ and k is the number of connected components in $\hat{C}$.

Hence, as a consequence of Theorem 1, we can introduce the constraint $\mathbf{rank}(L_{\hat{C}}) = n - k$ to ensure that $\hat{C}$ consists of k connected components. Unfortunately, as the rank function is non-convex, the optimization problem is difficult to solve. To solve this issue, we utilize the fact that the rank of a matrix corresponds to the number of non-zero eigenvalues. Therefore, the constraint $\mathbf{rank}(L_{\hat{C}}) = n - k$ can be approximated in terms of the k smallest eigenvalues of $L_{\hat{C}}$, $\lambda_1, ..., \lambda_k$, by the following minimization problem

$$\min \quad \alpha_1 \sum_{i=1}^{k} \lambda_i, \tag{2.37}$$

given a sufficiently large constant α_1 [7]. The idea is that if α_1 is sufficiently large, it will force the sum of the eigenvalues to go towards zero. To continue, according to [23] we have

$$\sum_{i=1}^{k} \lambda_i = \min_{F \in \mathcal{R}^{n \times k}, F^\intercal F = I} \mathbf{trace}(F^\intercal L_{\hat{C}} F). \tag{2.38}$$

By adding the constraint (2.38) to the optimization problem in (2.37) we get

$$\begin{aligned} \min \quad & ||\hat{C}||_* + \alpha_1 \cdot \mathbf{tr}(F^\intercal L_{\hat{C}} F) \\ \text{s.t.} \quad & A = \hat{C} + R + E, \quad \pi_\Omega(E) = 0, \\ & \hat{C} \geq 0, \quad R \geq 0, \\ & F \in \mathbb{R}^{n \times k}, F^\intercal F = I. \end{aligned} \tag{2.39}$$

The new, improved model (2.39) captures the underlying community structure more accurately than the previous model (2.37) as it now constructs $\hat{C}$ in such a way that it consists of the correct number of connected components. However, there is still room for improvement. One obvious flaw of our current model is that it focuses solely on constructing the perfect clustering matrix while ignoring many of the inherent properties of the noise matrix. To illustrate some of the current limitations we consider the following example. Assume that we have an initial network with no overlapping edges consisting of $k+1$ fully connected components and k communities. Let $N_1, ..., N_{k+1}$ denote the fully connected components listed in ascending order of size. In this scenario, the current model prioritizes the smaller components during the construction of $\hat{C}$ as the nuclear norm increases with the size of the fully connected component, whilst the rank of the Laplacian remains the same for all possible constructions. Therefore, the largest connected component is moved to the noise matrix. Intuitively, the smallest connected component should be considered to be noise, not the largest component. To fix the issue, we add the constraint $\alpha_2 \cdot ||R||_1$ that ensures that the number of noisy edges is minimal while still satisfying the constraints regarding $\hat{C}$. The final model is given by

$$\begin{aligned} \min \quad & ||\hat{C}||_* + \alpha_1 \cdot \mathbf{tr}(F^\intercal L_{\hat{C}} F) + \alpha_2 \cdot ||R||_1 \\ \text{s.t.} \quad & A = \hat{C} + R + E, \quad \pi_\Omega(E) = 0, \\ & \hat{C} \geq 0, \quad R \geq 0, \\ & F \in \mathbb{R}^{n \times k}, F^\intercal F = I. \end{aligned} \tag{2.40}$$

To solve (2.40), an iterative strategy called coordinate descent is adopted. Coordinate descent works by updating one variable at the time while keeping the other variables fixed. Begin with updating F. By fixing $\hat{C}$,R and E, only the second term in the objective function needs to be considered:

$$\begin{aligned} \min_F \quad & \alpha_1 \cdot \mathbf{tr}(F^\intercal L_{\hat{C}} F) \\ \text{s.t.} \quad & A = \hat{C} + R + E, \quad \pi_\Omega(E) = 0 \\ & \hat{C} \geq 0, \quad R \geq 0, \\ & F \in \mathcal{R}^{n \times k}, F^\intercal F = I. \end{aligned} \tag{2.41}$$

The optimal solution F to (2.41) is generated by the k eigenvectors of $L_{\hat{C}}$ corresponding to the k smallest eigenvalues.

Solving for $\hat{C}$ is more difficult as the objective function consists of two terms depending on $\hat{C}$, namely $||\hat{C}||_*$ and $\mathbf{tr}(F^\intercal L_{\hat{C}} F)$ To solve this, we adopt a convex optimization technique called *Alternating Direction Method of Multipliers* (ADMM). The first step of ADMM is to break up the problem into smaller pieces, which, by themselves, are easier to solve than the original problem [24]. To decompose the problem, we introduce an auxillary variable $\hat{Z} = \hat{C}$ which makes it possible to

rewrite (2.40) as the following:

$$\begin{aligned} \min_{\hat{C}} \quad & ||\hat{C}||_* + \alpha_1 \cdot \mathbf{tr}(F^\intercal L_{\hat{Z}} F) + \alpha_2 \cdot ||R||_1 \\ \text{s.t.} \quad & A = \hat{C} + R + E, \quad \pi_\Omega(E) = 0, \quad \hat{Z} = \hat{C} \\ & \hat{C} \geq 0, \quad \hat{Z} \geq 0, \quad R \geq 0, \\ & F \in \mathbb{R}^{n \times k}, F^\intercal F = I. \end{aligned} \tag{2.42}$$

The second step of ADMM is to reformulate the problem by using the augmented Lagrangian method. The augmented Lagrangian method incorporates the constraints into the objective function to form the following unconstrained objective:

$$\begin{aligned} \min_{\hat{C}} \quad & ||\hat{C}||_* + \alpha_1 \cdot \mathbf{tr}(F^\intercal L_{\hat{Z}} F) + \alpha_2 \cdot ||R||_1 + \mathbf{tr}(\lambda_1(\hat{Z} - \hat{C})) + \\ & \frac{\mu}{2}||\hat{Z} - \hat{C}||_F^2 + \mathbf{tr}(\lambda_2(A - \hat{C} - R - E)) + \frac{\mu}{2}||A - \hat{C} - R - E||_F^2, \end{aligned} \tag{2.43}$$

where λ_1 and λ_2 are Lagrange multipliers and $\mu > 0$ is a parameter. The advantage of expressing the original problem as (2.43) is that we can solve for $\hat{Z}$ and $\hat{C}$ separately. First, we solve for $\hat{Z}$ by fixing $F, \hat{C}$,R and E. As all variables except $\hat{Z}$ are fixed, only the terms involving $\hat{Z}$ have to be considered, that is

$$\min_{\hat{Z}} \quad \alpha_1 \cdot \mathbf{tr}(F^\intercal L_{\hat{Z}} F) + \mathbf{tr}(\lambda_1(\hat{Z} - \hat{C})) + \frac{\mu}{2}||\hat{Z} - \hat{C}||_F^2 \tag{2.44}$$

The solution to (2.44) is found by taking the derivative with respect to $\hat{Z}$ and computing for what value $\hat{Z}$ it is equal to zero, that is,

$$\frac{\partial}{\partial \hat{Z}} \left(\alpha_1 \cdot \mathbf{tr}(F^\intercal L_{\hat{Z}} F) + \mathbf{tr}(\lambda_1(\hat{Z} - \hat{C})) + \frac{\mu}{2}||\hat{Z} - \hat{C}||_F^2 \right) = 0 \tag{2.45}$$

For the rightmost term in (2.45), we have

$$\frac{\partial}{\partial \hat{Z}} \frac{\mu}{2}||\hat{Z} - \hat{C}||_F^2 = \frac{\partial}{\partial \hat{Z}} \frac{\mu}{2} \mathbf{tr}((Z - \hat{C})^\intercal (Z - \hat{C})) \tag{2.46}$$

$$= \frac{\partial}{\partial \hat{Z}} \frac{\mu}{2} (Z^\intercal \hat{Z} - \hat{Z}^\intercal \hat{C} - \hat{C}\hat{Z} - \hat{C}^\intercal \hat{C}) = \mu(\hat{Z}^\intercal - \hat{C}^\intercal). \tag{2.47}$$

To continue, for the middle term in (2.45), we have

$$\frac{\partial}{\partial \hat{Z}} \mathbf{tr}(\lambda_1(\hat{Z} - \hat{C})) = \lambda_1^\intercal. \tag{2.48}$$

Lastly, according to [7], for the leftmost term, we have

$$\frac{\partial}{\partial \hat{Z}} \alpha_1 \mathbf{tr}(F^\intercal L_{\hat{Z}} F) = H, \tag{2.49}$$

where $H = ||f_i - f_j||_F^2$ where f_i is the $i-th$ of $F \in \mathbf{R}^{n \times k}$. By putting (2.46), (2.48), (2.49) together, we get the closed-form of Z that solves (2.45), that is,

$$\hat{Z} = \hat{C} - \frac{\alpha_1 H + \lambda_1}{\mu}. \tag{2.50}$$

By fixing $F, \hat{Z}, R$, and E and utilizing Lagrange multipliers in order to relocate the constraint to the objective function, the clustering matrix $\hat{C}$ is updated by solving the following minimization problem:

$$\begin{aligned}\min_{\hat{C}} \quad & ||\hat{C}||_* + \mathbf{tr}(\lambda_1^\intercal(\hat{Z} - \hat{C})) + \frac{\mu}{2}||\hat{Z} - \hat{C}||_F^2 \\ & + \mathbf{tr}(\lambda_2^\intercal(A - \hat{C} - R - E)) + \frac{\mu}{2}||A - \hat{C} - R - E||_F^2\end{aligned} \tag{2.51}$$

According to [25], (2.51) can be rewritten as

$$\min_{\hat{C}} \quad \frac{1}{2\mu}||C||_* + \frac{1}{2}||\hat{C} - (\frac{1}{2\mu}(\lambda_1 + \lambda_2) + \frac{1}{2}(A + \hat{Z} - R - E))||_F^2. \tag{2.52}$$

The objective function (2.52) is a well-known objective function with the following solution

$$\hat{C} = D_{\frac{1}{2\mu}}((\frac{1}{2\mu}(\lambda_1 + \lambda_2) + \frac{1}{2}(A + \hat{Z} - R - E)), \tag{2.53}$$

where D_μ is the soft-thresholding operator D_μ defined as

$$D_\mu(X) = U \cdot D_\mu(\Sigma) \cdot V^\intercal, \tag{2.54}$$

where $D_\mu(\Sigma) = diag(\sigma_i - \mu)_+$.

The noise matrix R is updated by keeping $\hat{C}, \hat{Z}, E$, and F fixed and solving the following minimization problem:

$$\min_{R} \quad \alpha_2||R||_1 + \mathbf{tr}(\lambda_2^\intercal(A - \hat{C} - R - E)) + \frac{\mu}{2}||A - \hat{C} - R - E||_F^2 \tag{2.55}$$

The solution to (2.55) is given by

$$R = S_{\frac{\alpha_2}{\mu}}(\frac{\lambda_2}{\mu} + (A - \hat{C} - E)), \tag{2.56}$$

where S_μ is the shrinkage operator, that is,

$$S_\mu(x) = \begin{cases} x - \mu & \text{if} \quad x \geq \mu, \\ 0, & \text{otherwise.} \end{cases} \tag{2.57}$$

The supplement matrix E is updated by fixing the other variables $\hat{C}, R$, and F by solving the following optimization problem:

$$\min_{E} \quad \mathbf{tr}(\lambda_2^\intercal(A - \hat{C} - R - E)) + \frac{\mu}{2}||A - \hat{C} - R - E||_F^2 \tag{2.58}$$

The solution to (2.58) is given by

$$E = \pi_\Omega(A - \hat{C} - R + \frac{\lambda_2}{\mu}) \tag{2.59}$$

Lastly, the Lagrange multipliers are updated iteratively according to

$$\lambda_1 = \lambda_1 + \mu(\hat{Z} - \hat{C}), \tag{2.60}$$

$$\lambda_2 = \lambda_2 + \pi_\Omega(\mu(A - \hat{C} - R - E)). \tag{2.61}$$

By iteratively updating $\hat{C}, \hat{Z}, R, E$, and F a solution to (2.40) is obtained which consists of a Laplacian matrix $L_{\hat{C}}$ which is approximately block-diagonal with k blocks. Hence, the Laplacian matrix can be written on the form in (2.62).

$$L_{\hat{C}} \approx \begin{pmatrix} B_1 & & 0 \\ & B_2 & \\ 0 & & B_k \end{pmatrix} \tag{2.62}$$

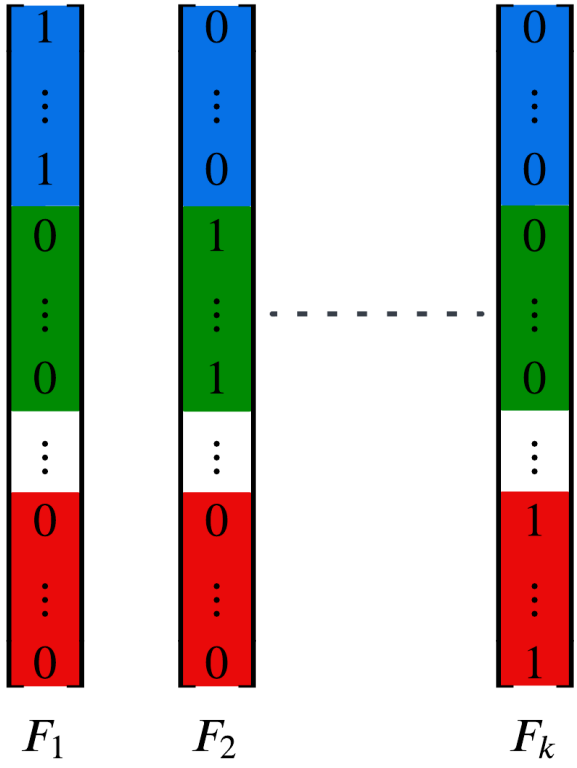


Figure 2.6: The strucutre of the spectral embedding F.

According to the properties of block diagonal matrices, the eigenvalues of $L_{\hat{C}}$ satisfy

$$det \begin{pmatrix} B_1 - \lambda I & 0 & \dots & 0 \\ 0 & B_2 - \lambda I & \dots & 0 \\ \vdots & & \ddots & \\ 0 & \dots & & B_k - \lambda I \end{pmatrix} = 0. \tag{2.63}$$

By utilizing the block matrix properties we have

$$det\begin{pmatrix} B_1 - \lambda I & 0 & \dots & 0 \\ 0 & B_2 - \lambda I & \dots & 0 \\ \vdots & & \ddots & \\ 0 & \dots & & B_k - \lambda I \end{pmatrix} = det(B_1 - \lambda I)det(B_2 - \lambda I) \tag{2.64}$$

$$...det(B_k - \lambda I) = 0. \tag{2.65}$$

Hence, the eigenvalues of the block matrices are also eigenvalues of $L_{\hat{C}}$. To continue, assume the block $B_2 \in \mathcal{R}^{m_1 \times m_1}$ has an eigenvector $\mathbf{v} = (v_1, ..., v_{m_1})$, then $\mathbf{w} = (0, \mathbf{v}, 0, ..., 0) \in \mathcal{R}^n$ is an eigenvector of $L_{\hat{C}}$ as

$$B_2\mathbf{v} = \lambda\mathbf{v} \Rightarrow L_{\hat{C}}\mathbf{w} = \mathbf{w}\lambda. \tag{2.66}$$

As each block $B_1, ..., B_k$ corresponds to a connected component, we have according to Theorem 1, that each block has an eigenvalue 0 with multiplicity 1.

Theorem 2: The eigenvalues of a Laplacian matrix L are non-negative.

According to theorem 2, all eigenvalues of $L_{\hat{C}}$ are non-negative. As the spectral embedding F is constructed to include the k eigenvectors corresponding to the k smallest eigenvalues of $L_{\hat{C}}$, we have according to theorems 1 and 2 that all the corresponding eigenvalues are 0 and that the spectral embedding is a span of k orthogonal eigenvectors on the form in Figure 2.6. Thus, to obtain the community structure, we treat F as a similarity matrix and run a simple k-means algorithm on F to obtain the community assignment for each node [26]. Putting it all together, we get the following algorithm in 2 for community detection in single-layered networks.

Algorithm 2 CLMC for Community Detection in Single-Layered Networks

Input: Adjacency matrix A, parameters α_1, α_2
Output: $\hat{C}, R, F$
Initialize $\hat{Z} = \hat{C} = \mathbf{0}, R = \mathbf{0}, E = \mathbf{0}, \lambda_1 = \mathbf{0}, \lambda_2 = \mathbf{0}, \mu = 10^-6, \mu_{\max} = 10^{10}, \rho = 1.5, \epsilon = 10^{-8}$
while not ($||A - \hat{C} - R - E||_F^2 < \epsilon$ and $||\hat{Z} - \hat{C}||_F^2$ **do**
    Update F by solving (2.41)
    Update $\hat{Z}$ by solving (2.44)
    Update $\hat{C}$ by solving (2.51)
    Update R by solving (2.55)
    Update E by solving (2.58)
    Update λ_1 according to (2.60)
    Update λ_2 according to (2.61)
    Update $\mu = \min(\rho \cdot mu, \mu_{\max})$
end while
Perform k-means on F to form k communities =0

2.2.3 Impact of parameters α_1 and α_2

The parameters α_1 and α_2 together serve as a balancing scale that can shift the focus of the algorithm in a given direction. By increasing α_1, the algorithm becomes increasingly focused on minimizing its corresponding factor, $\mathbf{tr}(F^\intercal L_{\hat{C}} F)$. In other words, α_1 can be viewed as a penalty parameter that penalizes the algorithm for not constructing a clustering matrix consisting of exactly k blocks. However, if α_1 becomes too large in relation to α_2, then classifying connections as noise goes relatively unpunished and the algorithm tends to classify too many edges as noise. The amount of edges classified as noise is regulated by the parameter α_2. By increasing α_2 we instead shift the focus of the algorithm such that it is required to find a low-rank solution such that the number of edges in the noise matrix is kept at a reasonable level. The choice of parameters α_1 and α_2 is crucial for obtaining good performance; thus, in Sections 2.4 and 3.3 we describe in detail how to tune them.

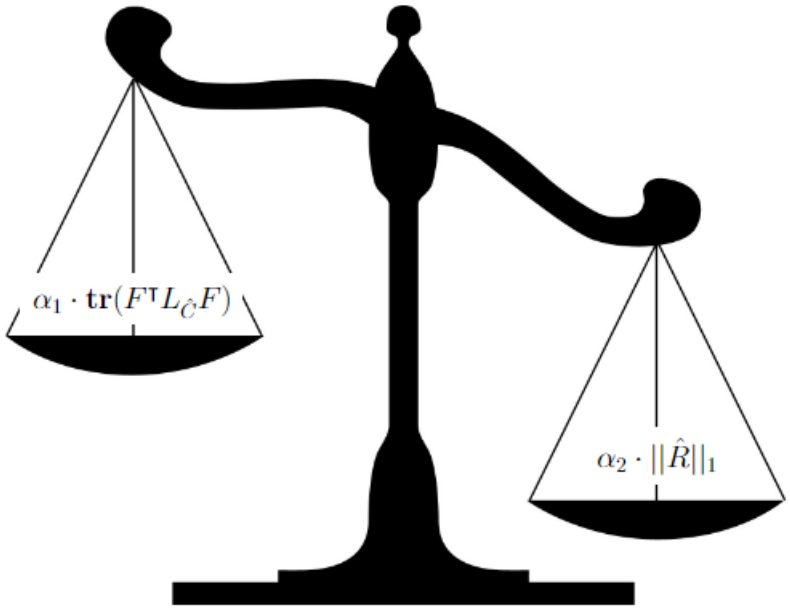


Figure 2.7: A depiction of the relation between the parameters α_1 and α_2.

2.3 Community detection in multiplex networks

2.3.1 Multiplex networks

In conventional monolayered network theory, a network is represented by a graph $G = (V, E)$, where V is a set of nodes and E the connections between nodes. The relations in the network, represented by its edges, are binary. A pair of nodes is either disconnected or connected by an edge. In reality, there may be multiple different relations between nodes. For instance, in the case of social network analysis, there may be several different types of relations between people. They could be married or related through a sports organization, or simply share a love of chocolate. In Figure 2.8 we see an example of such a multi-relation network. To accommodate multi-

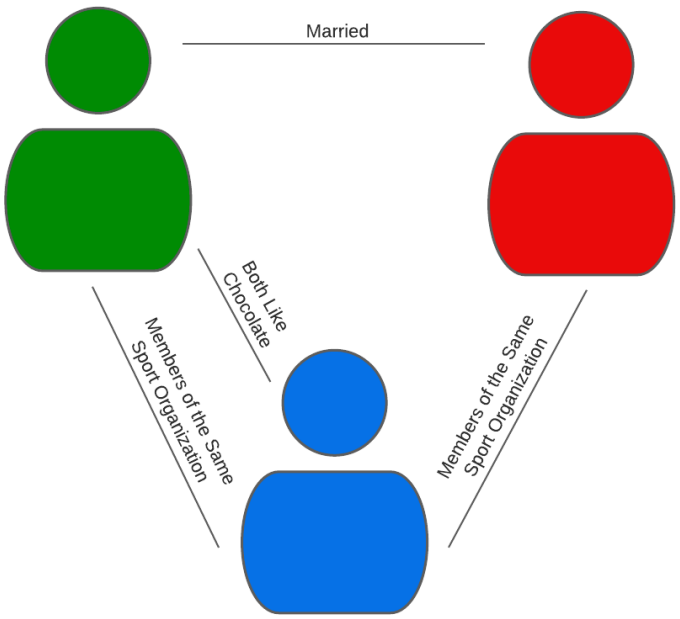


Figure 2.8: A multi-relation network consisting of people related through different relations.

ple relations within networks, we introduce the data structure known as multiplex networks. A multiplex network is a collection of layers $W = \{G(l) = (V, E(l))\}_{l=1}^{L}$ where each layers, $G(l)$, is a graph representing one of the L relations within the network. Each layer consists of the same set of nodes, V. The representation of a given node $v \in V$ in layer l is denoted by $v(l)$ and the set of representations representing the same node $v \in V$, $a_v = \{v(l)\}_{l=1}^{L}$ is referred to as an actor. Let $\{R(l)\}_{l=1}^{L}$ denote the set of relations in the network, and then $v(l)$ and $w(l)$ is said to be related through relation $R(l)$ if and only if $e = (v(l), w(l)) \in E_l$. Edges of this type, that connects two nodes in the same layer are called intralayered connections. In addition to the *intralayered connections*, there is a second type of connection called interlayered connections which constitute the edges between node representations in different layers [27]. In Figure 2.9 we see a simple multiplex network

consisting of two layers, where the different types of connections, both intralayered and interlayered connections, are displayed.

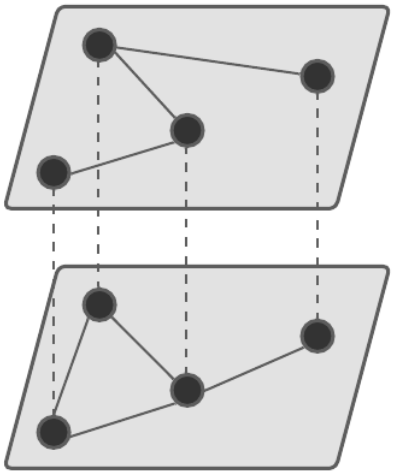


Figure 2.9: A two-layered multiplex network. The dotted lines and solid lines represent interlayed connections and intralayered connections respectively.

2.3.2 Community structures in multiplex networks

In single-layered networks, the definition of a community structure is rather straightforward. The community labels are simply handed out based on the cluster to which a given node is most densely connected. The essence of community detection remains the same in multiplex networks as in single-layered networks. However, there are a few details that need to be addressed before we can extend the subject of community detection to multiplex networks. In the case of multiplex networks, the definition of what constitutes a community structure is not trivial; in fact, there are several viable definitions for community structures in multiplex networks. One possible definition is to give each actor a single community label [5]. In other words, the nodes in each layer that represent the same actor are given the same community label. We refer to this type of community structure as a *single-actor community structure*. The second option is to allow for a more complex community structure that allows the nodes representing the same actor to belong to different communities, under some restrictions. The idea, introduced in [8], is that the appearance of a given community in the layer is binary, either it is present in a given layer or it is not. If a given community is present in two different layers l_1 and l_2, then it refers to the same set of nodes, enforcing consistent communities to emerge that can easily be fused together. This community structure constraint is called the *well-partioned property* (WPP) and is more precisely defined as follows. A given community structure, C, satisfy the WPP if and only if for each pair of node representations (v_i, v_j) with community labels $t_i(l) = \alpha$ and $t_j(l) = \beta$ satisfy the following:

$$\text{If} \quad \alpha = \beta, \quad \text{then} \begin{cases} t_i(l') = t_j(l') = \alpha \\ \text{or} \begin{cases} t_i(l') \neq \alpha \\ t_j(l') \neq \alpha \end{cases} \end{cases},$$

$$\text{If} \quad \alpha \neq \beta, \quad \text{then} \begin{cases} t_i(l') \neq \beta \\ t_j(l') \neq \alpha \end{cases}.$$

We refer to a community structure that satsify WPP as a *complex community structure.*

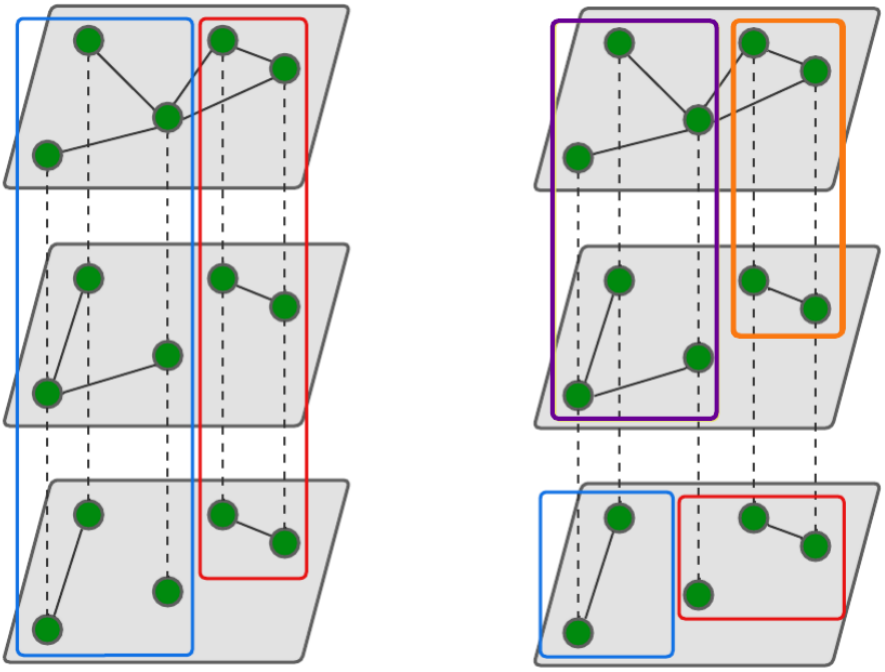


Figure 2.10: An example of a single-actor community structure (left) and an example of a complex community structure (right).

2.3.3 Community detection in multiplex networks

Here, the problem of community detection is defined for multiplex networks. As previously mentioned, there are multiple definitions for community structures in multiplex networks. Therefore, the problem of community detection is defined separately for single-actor and complex community structures. Given a multiplex network, $W = \{G = (V, E(l))\}_{l=1}^{L}$, community detection aims to determine a community structure $C = \{C_1(l), ..., C_{k(l)}(l)\}_{l=1}^{L}$ which divides the node representations in

each layer into $k(l)$ communities that are densely intraconnected with few overlaps between distinct community members. For single-actor community detection the number of communities in each layers is same, that is, $k(l_1) = ... = k(l_L)$ and the layer partitions are equivalent across all layers. In the case of complex community detection the number of communities is not required to be the same in every layer, however, the layer partitions must adhere to the WPP.

2.3.4 Generalization of SBM

The stochastic block model defined in Section 2.2.1.1 can be generalized to multiplex networks such that it is able to generate multiplex networks with either single-actor or complex community structures. A multiplex SBM, mSBM(C, T), consists of a set of block matrices $C = \{C(l)\}_{l=1}^{L}$ and a set of node labels $T = \{T(l)\}_{l=1}^{L}$, one for each layer $l \leq L$. Hence, a multiplex SBM can be thought of as a set of SBM's, that is, mSBM $= \{SBM_l(C(l), T(l))\}_{l=1}^{L}$ [28]. In order to generate a single-actor community structure, the block matrices and node labels are identical in all layers, in other words, $T(l_1) = ... = T(l_L)$. A multiplex network with a complex community structure is generated by choosing different block matrices for each layer and node labels that satisfy the WPP constraint [8].

2.3.5 Extension techniques for extending mono-layered community detection methods to multiplex networks

In this section, three different techniques for extending single-layered community detection methods are described.

2.3.5.1 Flattening

The simplest extension of mono-layered community detection methods involves the transformation of a multiplex network into a simple monolayered network that inherits some of the properties of the layers in the original network. A so-called *flattening* algorithm is utilized to perform the conversion by projecting the edges of each layer onto a single layer (see Figure 2.11). The main advantage of flattening is that it enables the use of conventional monolayered community detection on the flattened network [29]. However, there are a few possible drawbacks to its straightforward design. The main issue with flattening is that noise can easily affect performance. In fact, it can magnify the impact of noisy edges as they are now present throughout the entire flattened network rather than being confined to a single layer of the multiplex network. The drawback can be somewhat mitigated by introducing some constraints to the flattening algorithm [30]. Before an edge is projected onto the flattened network, it undergoes a filtering process in which the number of its occurrences in distinct layers within the original multiplex network is considered. The idea is that noisy edges only appear in a few layers, while crucial components of the community structure appear in the majority of layers. Therefore, it can be useful to define a threshold for deciding whether or not an edge should be included in the flattened network. While the introduction of weights can improve performance by removing noise, it can also create a bias towards edges appearing in several layers

[5]. The algorithm for the flattening technique is presented in algorithm 3 and let f-CLMC and f-BP refer to the flattened versions of CLMC and BP.

Algorithm 3 Flattened community detection in multiplex networks

Input: Multiplex network $W = \{G(l) = (V, E(l))\}_{l=1}^{L}$, an algorithm $\mathcal{A}(\tau)$ with parameters τ, threshold θ
Output: Community structure $\mathcal{C}$
Initialize: Flattened network $A_f = \mathbf{0}$
for each pair of actors (i,j) **do**
  if $\sum_l (i(l), j(l)) \geq \theta$ **then**
    $A_f(i, j) = 1$
  end if
end for
Run the single-layered community detection algorithm $\mathcal{A}(\tau)$ on A_f to obtain the community structure $\mathcal{C}$
Return: $\mathcal{C}$

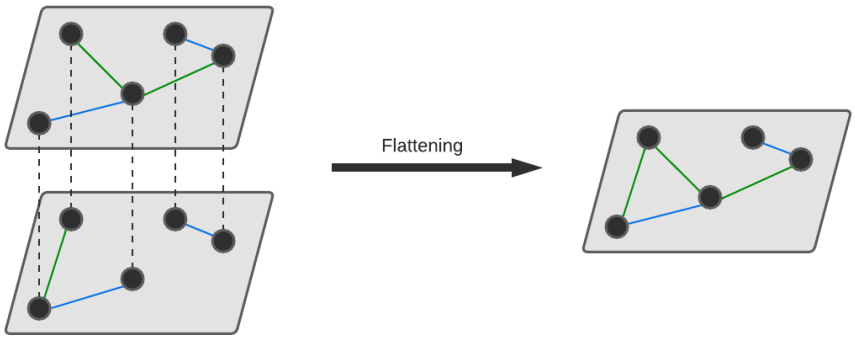


Figure 2.11: A visualization of the flattening technique for flattening a multiplex network. Blue edges signify edges that are present in both layers, and green edges are edges present in only one of the layer.

2.3.5.2 Layer-by-layer

By combining layers in a systematic way to form a single-layered network, the flattening approach finds a consensus community for the entire multiplex network. A different approach is to instead combine the resulting community structures by separately running a community detection algorithm on each individual layer. We call this class of algorithms *Layer by Layer* algorithms, and the setup for a layer by layer algorithm is as follows: Given a multiplex network $W = \{G(l) = (V, E(l))\}_{l=1}^{L}$ we run a community detection algorithm on each layer $l = 1, ..., L$ to decompose its corresponding community structure $\mathcal{C}(l)$. The set of these community structures, $\mathcal{E} = \{\mathcal{C}(l)\}_{l=1}^{L}$, is called an ensemble. To continue, the algorithm outputs a consensus community structure $\mathcal{C}^*$ which is representative of the ensemble $\mathcal{E}$. There are many possibilities for how you could form a consensus community structure given an ensemble [31], [32].

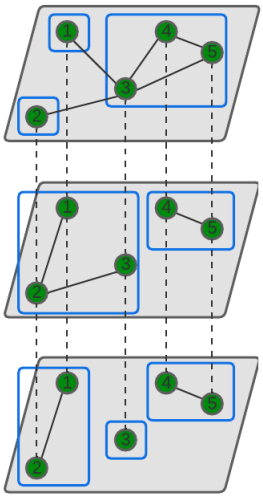
A consensus can only be reached if there is some general agreement or similarity between the different community structures. Therefore, to construct a consensus community structure, we first formulate a way of measuring the degree of similarity between distinct community structures. Given a multiplex network $W = \{G(l) = (V, E(l))\}_{l=1}^{L}$ and an ensemble of communities $\mathcal{E}$ for W, we define the consensus matrix M as an $n \times n-$matrix with elements

$$M_{ij} = \sum_{l} m(i, j, l), \tag{2.67}$$

where

$$m(i, j, l) = \begin{cases} 1 & \text{if} \quad v_i(l), v_j(l) \in C_q(l), \quad C_q(l) \in \mathcal{C}(l), \\ 0 & \text{otherwise.} \end{cases} \tag{2.68}$$

In other words, the $i, j-$th element of the consensus matrix corresponds to the fraction of community structures in the ensemble where the nodes v_i and v_j belong to the same community. In Figure 2.12 we see an example of a three-layered multiplex network with its layer-specific community structure and corresponding consensus matrix. Similar to the flattening technique, we apply a filter for the consensus matrix, where the elements of the consensus matrix are set to zero if they are below a given threshold θ. Finally, the consensus community structure is constructed by a simple breadth-first search starting at each node on the consensus matrix that has undergone a filtering process with a given threshold θ. In Figure 2.12 we obtain the consensus community structure $\mathcal{C}^* = \{\{1, 2\}, \{3\}, \{4, 5\}\}$ given a filtering threshold constant $\theta = 2$. The algorithm for the layer-by-layer technique is presented in algorithm 4 and let LbL-CLMC and LbL-BP refer to the layer-by-layer multiplex extensions of CLMC and BP.



$$M = \begin{pmatrix} 0 & 2 & 1 & 0 & 0 \\ 2 & 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 3 & 0 \end{pmatrix}$$

Figure 2.12: An example of the resulting consensus matrix produced by the layer-by-layer technique.

Algorithm 4 Layer-by-Layer Community Detection in Multiplex Networks

Input: Multiplex network $W = \{G(l) = (V, E(l))\}_{l=1}^{L}$, an algorithm $\mathcal{A}(\tau)$ with parameters τ, threshold θ
Output: Consensus community structure $\mathcal{C}^*$
Initialize: Consensus matrix $M = \mathbf{0}$
for every layer l **do**
  Run the algorithm $\mathcal{A}(\tau)$ on layer l to construct the community structure $\mathcal{C}(l)$
end for
Form the ensemble $\mathcal{E} = \{\mathcal{C}(l)\}_{l=1}^{L}$
for every community structure $\mathcal{C}(l)$ in $\mathcal{E}$ **do**
  for every community C_q in $\mathcal{C}(l)$ **do**
    for every pair of nodes $(v_i(l), v_j(l))$ **do**
      $M_{ij} = M_{ij} + m(i, j, l)$
    end for
  end for
end for
Filter M according to the threshold θ
for every unvisited node i **do**
  Run BFS starting at node i to form the community C_q
end for
Return: Consensus community structure $\mathcal{C}^* = \{C_1, ..., C_k\}$, where k is the number of communities found during the BFS.

2.3.5.3 Global algorithms

The main limitation of the flattening and layer-by-layer techniques is that, by definition, they can only find overlapping-actor communities and fail to find more complex community structures that satisfy the well partitioned property. These complex community structures can only be found using community detection methods that directly process the multiplex structure. In Section 2.3.5.3.1, an extension of belief propagation to the case of multiplex networks that work directly on the multiplex network itself is described.

The plan was originally, if time permitted, to formulate a generalization of RMP that works directly on the multiplex network itself. However, during our research process such an extension was publicized by [33]. Hence, we decided to try to generalize a somewhat similar method called dictionary learning to detect communities in multiplex networks. However, as the time was starting to run out we will not include these results as the algorithm is still in development and we will instead work on an independent publication.

2.3.5.3.1 Generalization of belief propagation to multiplex networks

Here we present an extension of the BP algorithm, in accordance with [8] that operate directly on the global multilayered network in order to enable detection of complex community structure. The extension is similar to the original single-layered

algorithm, but with some key differences including an additional interlayered factor node that enforces the WPP constraint and slightly adjusted messages that include information from nodes in neighboring layers.

Similar to the single-layered case, community detection in multiplex networks can naturally be expressed as a Bayesian inference problem where the goal is to find the community labels $\{T(l)\}_{l=1}^{L}$ most likely to have generated a given multiplex network $W = \{G(l) = (V, E(l))\}_{l=1}^{L}$. That is

$$\begin{aligned}\{T_{\text{MAP}}(l)\}_{l=1}^{L} &= \arg\max_{\{T(l)\}_{l=1}^{L}} P(\{T(l)\}_{l=1}^{L}|\{G(l)\}_{l=1}^{L}, \theta) \\ &= \arg\max_{\{T(l)\}_{l=1}^{L}} \frac{P(\{T(l)\}_{l=1}^{L}, \{G(l)\}_{l=1}^{L}|\theta)}{\sum_{\{t_i(l)\}} P(\{t_i(l)\}, \{G(l)\}_{l=1}^{L}|\theta)}.\end{aligned} \tag{2.69}$$

By applying Baye's rules, the denominator in the right hand side of (2.69) can be expressed as:

$$\begin{aligned}P(\{T(l)\}_{l=1}^{L}, \{G(l)\}_{l=1}^{L}|\theta) &= P(\{G(l)\}_{l=1}^{L}|\{T(l)\}_{l=1}^{L}, \theta) \cdot P(\{T(l)\}_{l=1}^{L}|\theta) \\ &= P(\{T(l)\}_{l=1}^{L}|\theta) \cdot \prod_{l=1}^{L} P(G(l)|T(l), \theta).\end{aligned} \tag{2.70}$$

The likelihood function $P(\{W(l)\}_{l=1}^{L}|\{T(l)\}_{l=1}^{L}, \theta)$ can be thought of as a series of events between each intralayered pair of nodes which can be expressed as:

$$\begin{aligned}P(\{G(l)\}_{l=1}^{L}|\{T(l)\}_{l=1}^{L}, \theta) = \frac{1}{Z} \prod_{l} \prod_{1<i<j<n} &(T_p(l)^{\intercal} C(l) T_q(l))^{A_{ij}(l)} \\ &\times (1 - T_p(l)^{\intercal} C(l) T_q(l))^{1-A_{ij}} \cdot \prod_{i(l)} n_{t_i}(l).\end{aligned} \tag{2.71}$$

To continue, the marginal distributions of the node representation $v_i(l)$ is given by

$$p_{i,l}(\tau) = P(t_i(l) = \tau|W, \theta) \sum_{\{t_j(l)\}: t_j(l)=\tau} P(T(l)|W, \theta). \tag{2.72}$$

The MAAP estimate for node i in layer l is given by

$$\hat{t}_{MAAP}(l) = \arg\max_{\tau} p_{i,l}(\tau). \tag{2.73}$$

However, this model is quite naive as it makes no effort to enforce the WPP constraints. In [8] an indicator function f_{WPP} is added to the model that enforces the WPP constraint defined as :

$$f_{WPP}(v_i(l), v_j(l), v_i(l'), v_j(l')) = \begin{cases} 1, & \text{if} \quad v_i(l), v_j(l), v_i(l'), v_j(l') \\ & \text{satisfy WPP}, \\ 0, & \text{otherwise.} \end{cases} \tag{2.74}$$

By multiplying (2.71) by (2.74) we get the modified likelihood function

$$\begin{aligned}P(W|\{T(l)\}_{l=1}^{L}, \theta) = \frac{1}{Z} \prod_{(i,j),(l,l')} f_{WPP}(v_i(l), v_j(l), v_i(l'), v_j(l')) \times \\ \left[\prod_{l} \left(\prod_{1<i<j<n} (T_p(l)^{\intercal} C(l) T_q(l))^{A_{ij}(l)} \cdot (1 - T_p(l)^{\intercal} C(l) T_q(l))^{1-A_{ij}} \cdot \prod_{i(l)} n_{t_i}(l) \right) \right].\end{aligned} \tag{2.75}$$

The modified likelihood function in (2.75) prioritizes labels that satisfy the WPP constraint as it becomes zero if it is not satisfied.

The groundwork for formulating the BP algorithm for multiplex has now been laid out. The objective is clear: we want to estimate the marginal probabilities $p_{i,l}(\tau)$ for each node $v_i(l)$ in the multiplex network. This is accomplished by first constructing a factor graph consisting of two types of factor nodes: intra-layered and inter-layered factor nodes. The intra-layered factor nodes are identical to the factor nodes in the single-layered case, located between each pair of nodes in the same layer and are defined as:

$$f(x_i(l), x_j(l)) = \begin{cases} c_{t_i(l),t_j(l)} & \text{if} \quad (i,j) \in E(l), \\ 1 - c_{t_i(l),t_j(l)} & \text{if} \quad (i,j) \notin E(l). \end{cases} \tag{2.76}$$

The second type of factor node is located between each 4-tuple on the form $(v_i(l), v_j(l)$ $,v_i(l'), v_j(l'))$ and is defined as $f_{WPP}(v_i(l), v_j(l), v_i(l'), v_j(l'))$. In Figure 2.13 the basic layout of the factor graph is visualized.

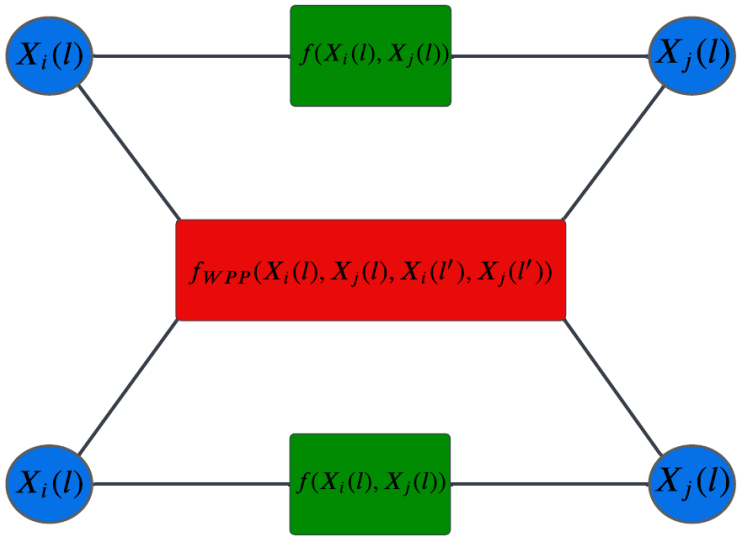


Figure 2.13: The layout of the multiplex version of BP.

In order to estimate the marginal probabilities $p_{i,l}(\tau)$ we run belief propagation on the proposed factor graph. Unfortunately, as the proposed model includes factor nodes that connect more than two variable nodes, we can no longer define direct messages between variables. Instead we define three types of messages: messages from factor nodes to variable nodes, intra-layered messages from variable nodes to factor nodes and finally inter-layered messages from variable nodes to factor nodes. The messages from factor nodes to variable nodes are defined as:

$$m_{a \to i}(x_i) = \sum_{\mathbf{x}_a \setminus x_i} f_a(\mathbf{x}_a) \cdot \prod_{j \in N(a) \setminus i} m_{j \to a}(x_j). \tag{2.77}$$

To continue, the intra-layered messages from variable nodes to intra-layered factor nodes are defined as:

$$m_{i\to a}^{t_i}(l) = \frac{1}{Z_{i\to a}(l)} n_{t_i}(l) e^{-h_{t_i}(l)} \prod_{d\in N_{intra}(i(l))\setminus a} \left(\sum_{t_d} c_{t_d,t_i(l)} m_{d\to i(l)}^{t_d} \right) \times \prod_{c\in N_{inter}(i)} \left(\sum_{t_j(l),t_i(l'),t_j(l')} f_{WPP}(t_i(l),t_j(l),t_i(l'),t_j(l') \cdot \prod_{k\in N_{inter}(c)\setminus i} m_{k\to c}^{t_k} \right), \tag{2.78}$$

where the external field is given by

$$h_{t_i}(l) = \frac{1}{N} \sum_k \sum_{t_k} c_{t_k t_i}(l) b_{t_k}^k(l). \tag{2.79}$$

Here $N_{intra}(i(l))$ and $N_{inter}(i)$ refer to the set of intra-layered neighbours and inter-layered neighbours to $i(l)$ respectively. The inter-layered messages from variable nodes to inter-layered factor nodes are defined as:

$$m_{i\to c}^{t_i} = \frac{1}{Z_{i\to c}} n_{t_i}(l) e^{-h_{t_i}(l)} \prod_{d\in N_{intra}(i)} \left(\sum_{t_d} c_{t_d,t_i} m_{d\to i}^{t_d} \right) \times \prod_{c^*\in N_{inter}(i)\setminus c} \left(\sum_{t_j(l),t_i(l'),t_j(l')} f_{WPP}(t_i(l),t_j(l),t_i(l'),t_j(l')) \cdot \prod_{k\in N_{inter}(c^*)\setminus i} m_{k\to c^*} \right). \tag{2.80}$$

The beliefs are given by

$$b_i^{t_i}(l) = \frac{1}{Z_i(l)} n_{t_i]}(l) e^{-h_{t_i}(l)} \prod_{d\in N_{intra(i(l))}} \left(\sum_{t_d} c_{t_d,t_i(l)} m_{d\to i(l)}^{t_d} \right) \times \prod_{c\in N_{inter}(i(l))} \left(\sum_{t_j(l),t_i(l'),t_j(l')} f_{WPP}(t_i(l),t_j(l),t_i(l'),t_j(l')) \cdot \prod_{k\in N_{inter}(c)\setminus i(l)} m_{k\to c}^{t_k} \right). \tag{2.81}$$

Lastly, the community label of each node representation is chosen according to the largest component of its belief, that is,

$$q_i(l) = \arg\max_{q_i(l)} b_i^{q_i}(l). \tag{2.82}$$

The algorithm for the multiplex version of BP (Multi-BP) is presented in algorithm 5.

Algorithm 5 Multi-BP for Community Detection for Multiplex Networks

Input: Multiplex network W, block matrix C, community sizes n_i, number of communities q, convergence parameter ϵ, maximum number of iterations $t_{\max}$
Output: Community labels
Initialize: Random normalized beliefs $b_i(l)$, external fields $h(l)$, messages $m_{i\to j}(l)$ $i = 1, ..., n$, $j \in N(i)$, $l = 1, ..., L$
while not conv $= |\sum m_{\text{new}} - m_{\text{old}}| < \epsilon$ and $t < t_{\max}$ **do**
  for every layer l **do**
    Pick a random ordering,
    for every directed edge $i \to j$ in layer l (in a random order) **do**
      Update messages $m_{i\to j}(l)$ according to (2.77)
      Update messages $m_{i\to c}$ according to (2.78)
      Update all $q-$components of $b_j(l)$ according to (2.81)
    end for
    Update external field $h(l)$ according to (2.79)
  end for
  conv $= |\sum m_{\text{new}} - m_{\text{old}}|$
  for every ordered pair of layers (l,l') **do**
    for every ordered pair of nodes (i,j) **do**
      Update message from $i \in l$ to the factor node between i and j, $m_{i\to c}(l)$ according to (2.80)
    end for
  end for
end while
Return: Community labels $q_i(l) = \arg\max_q b_i^q(l)$, $i = 1, ..., n$ and $1 \le l \le L$

2.4 Parameter tuning

In this section, the necessary theory required for parameter tuning is described. Firstly, in Section 2.4.1, the evaluation metric that determines the performance of a chosen set of parameters is described. In Sections 2.4.2 and 2.4.3, a procedure for finding the optimal parameters that minimize the evaluation metric is outlined.

2.4.1 The generalization error

The generalized error is a common metric to estimate the performance of a model and its capability to detect community structures in networks not involved in the training process. In other words, the generalized error is a metric for estimating how well a model generalizes to unseen data. The generalized error is defined in [34] as:

$$\mathbb{E}_{x_{\mathcal{T}} \sim \mathcal{G}_x} \left[E_{\mathcal{V}} \left(\mathcal{A}_\theta(\mathcal{T}) \right) \right], \tag{2.83}$$

where $\mathcal{A}$ denotes an algorithm (for example BP and CLMC) with a given set of parameters θ that was determined during the training process on the training set $\mathcal{T}$. To continue, $x_{\mathcal{T}} \in \mathcal{T}$ is a sample of random variables sampled from an underlying probability distribution ,$\mathcal{G}_x$, and, in the case of community detection, $\mathcal{G}_x$ is the underlying SBM. In Section 2.4.3 a parameter tuning technique is presented that utilizes a combination of cross-validation and grid search to find the parameters that minimize the generalized error.

2.4.2 Cross-validation

Cross-validation is a two-step validation method for assessing how effectively a particular model generalizes to independent data not included in the training process. The first step of cross-validation is to sample the observed data into two buckets: training data and validation data. More precisely, the observed data set $\mathcal{D} = \{x_1, ..., x_n\}$ is partitioned into $k \geq 2$ subsets $\mathcal{D}_1, \mathcal{D}_2, ..., \mathcal{D}_k$ such that they create a partition of $\mathcal{D}$ where $|\mathcal{D}_1| \approx |\mathcal{D}_2| \approx ... \approx |\mathcal{D}_k|$. To continue, the training set $\mathcal{T}_j$ and the validation set $\mathcal{V}_j$ are systematically chosen through k distinct set-ups on the following form

$$\mathcal{V}_j = \mathcal{D}_j, \quad \mathcal{T}_j = \mathcal{D} \setminus \mathcal{D}_j \quad \text{where} \quad j = 1, ..., k. \tag{2.84}$$

Figure 2.14 shows a set-up on the form in (2.84).

Cross-validation consists of a total of k training procedures, where each subset serves as the validation set **once** and is allocated to the training set for the remaining training procedures. During each training process, with the help of a community detection algorithm $\mathcal{A}$, the hyperparameters are trained on the training set and is then tried and tested on the validation data in order to determine how well it is able to find a community structure in the validation data that was not involved in the training process. As the parameters are chosen independently of the validation dataset it is possible according to [35] to estimate the generalized error as follows:

$$\mathbb{E}_{x_{\mathcal{T}} \sim \mathcal{G}_x} \left[E_{\mathcal{V}} \left((\mathcal{A}_\theta(\mathcal{T})) \right] \approx \frac{1}{k} \sum_{j=1}^{k} E_{\mathcal{V}_j} \left(\mathcal{A}_\theta(\mathcal{T}) \right), \tag{2.85}$$

where $E_{\mathcal{V}_j}$ is an error function that quantify to what degree the algorithm finds the correct community structure in the validation data.

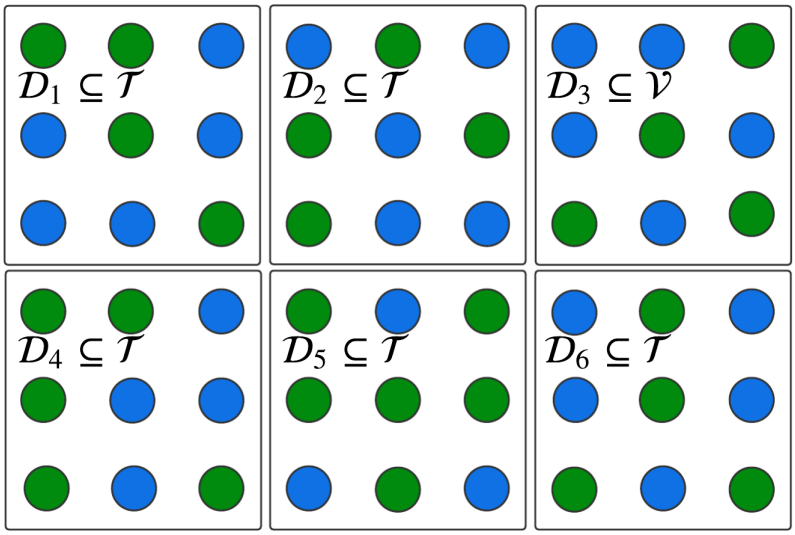


Figure 2.14: A partition on the form in (2.84).

2.4.3 Grid search

Grid search is a simple parameter tuning technique that performs an exhaustive search through a discrete set of predetermined parameter values in order to find the set of parameters that yield optimal performance [34]. In [36], grid search is combined together with cross-validation to find the parameter values that perform best on unseen data. Let Θ denote the set of predetermined parameter values, chosen by the user. For each set of parameters $\theta \in \Theta$, a model is trained and the generalized error is estimated with the help of cross-validation as explained in Section 2.4.2. In accordance with [37], the final parameters are selected to provide the smallest generalized error, that is,

$$\theta^* \approx \arg\min_{\theta \in \tilde{\Theta}} \frac{1}{k} \sum_{j=1}^{k} E_{\mathcal{V}_j} \left(\mathcal{A}_\theta(\mathcal{T}) \right). \tag{2.86}$$

As grid search estimates the generalized error for a predetermined set of parameters it is unlikely to find the theoretical optimal parameters. However, according to [34] it is in most cases plausible to find sufficiently good choices of parameters such that the error for unseen data is relatively low.

3

Methods

In this chapter, the data generation process and the methods used to determine the performance of our models are outlined.

3.1 Data

3.1.1 Synthetic networks

Unfortunately, at this moment of time, the number of publicly available multiplex networks are few and a well-established benchmark is yet undetermined. Therefore, to get a holistic understanding of the discussed algorithms, they are tried and tested on synthetic networks, in addition to real-world networks. Generating these synthetic networks is a rather straightforward process. As shown in Sections 2.2.1.1 and 2.3.4, the SBM and mSBM are natural generative models for generating both simple mono-layered networks and multiplex networks that inherit a community structure. The main benefit of utilizing SBM and mSBM as generative models is that different properties such as number of overlaps, average node degree, sparsity, degree of inter-connectivity, number of communities, and number of layers can be isolated in order to study their effect on performance. We isolate the community structure properties in the following manner. First, we determine a default mSBM for generating the synthetic networks. To continue, we change one of the parameters of the default mSBM at a time in order to study its impact on performance. Let mSBM($\{C(l)\}_{l=1}^{L}, \{T(l)\}_{l=1}^{L}$) with $L = 2$, $N(l) = 60$, $N_1(l) = 30$, $N_2(l) = 30\}$ for $l = 1, 2$ and

$$C(l) = \begin{pmatrix} c_{in}(l) & c_{out}(l) \\ c_{out}(l) & c_{in}(l) \end{pmatrix} = \begin{pmatrix} 0.9 & \frac{0.9}{0.3} \\ \frac{0.9}{0.3} & 0.9 \end{pmatrix}, \quad \text{for} \quad l = 1, 2, \tag{3.1}$$

be the default generative model. Hence, the default generative model generates multiplex networks consisting of two layers, 60 actors and two communities of equal size, that has on average an intraconnectivity of 90% and $\frac{0.9}{0.3}$% overlaps. This default generative model was chosen for two reasons. Firstly, multiplex networks generated by the default generative model consist of sufficiently large communities which should be detectable by the discussed community detection methods. In addition, the generated networks are not too large, which let us run a large number of tests in a reasonable amount of time.

3.1.2 Real-world networks

By utilizing the default generative model, it is possible to create community structures with very specific properties. However, as the synthetic networks are generated in a controlled environment, they do not necessarily capture the complexities of real-world community structures. Therefore, the performance of the discussed community detection methods is evaluated for two real-world multiplex networks, namely *C.Elegans* and *CKM*, in addition to the synthetically generated networks. These networks were chosen for three reasons. Firstly, the network profiles of C.Elegans and CKM are vastly different, both in its number of communities and noise profile. C.Elegans consist 279 nodes, 3 layers and 12 communities with a large number of overlaps. CKM consist of 246 nodes, 3 layers and 4 communities with a small number of overlaps. Secondly, in [33] a number of different community detection methods have been run on these networks, which provides a benchmark that the performance of our discussed community detection methods can be compared to. The Figures 3.1 and 3.2 shows the network profiles for C.ELEGANS and CKM respectively. The CKM and C.Elegans networks are accessible through [38] and [39] respectively.

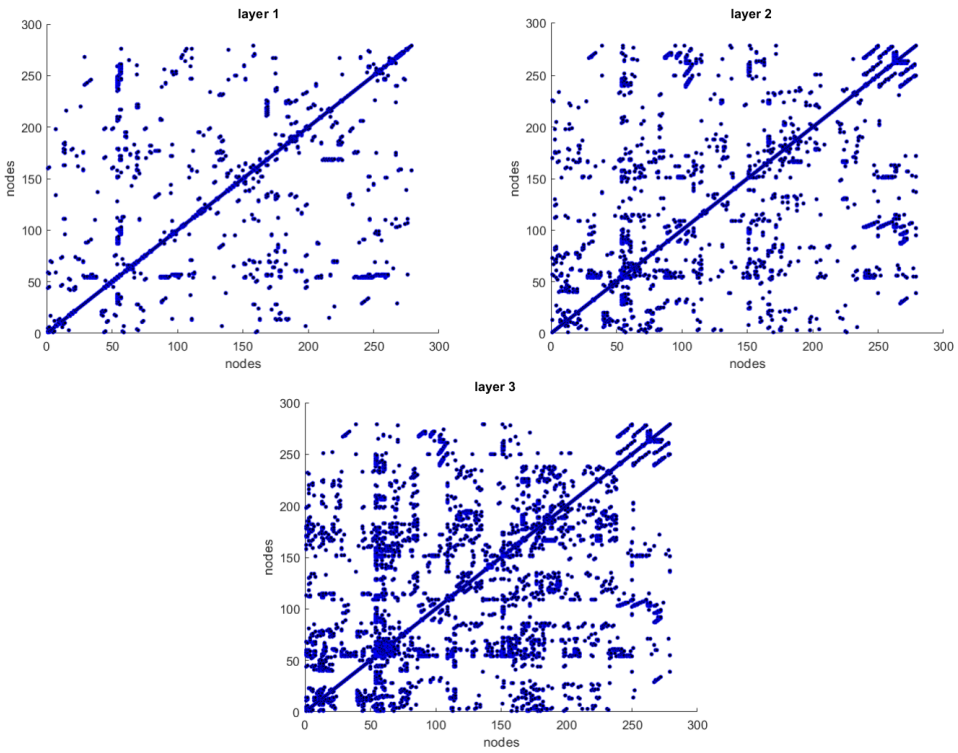


Figure 3.1: The layers of C.ELEGANS.

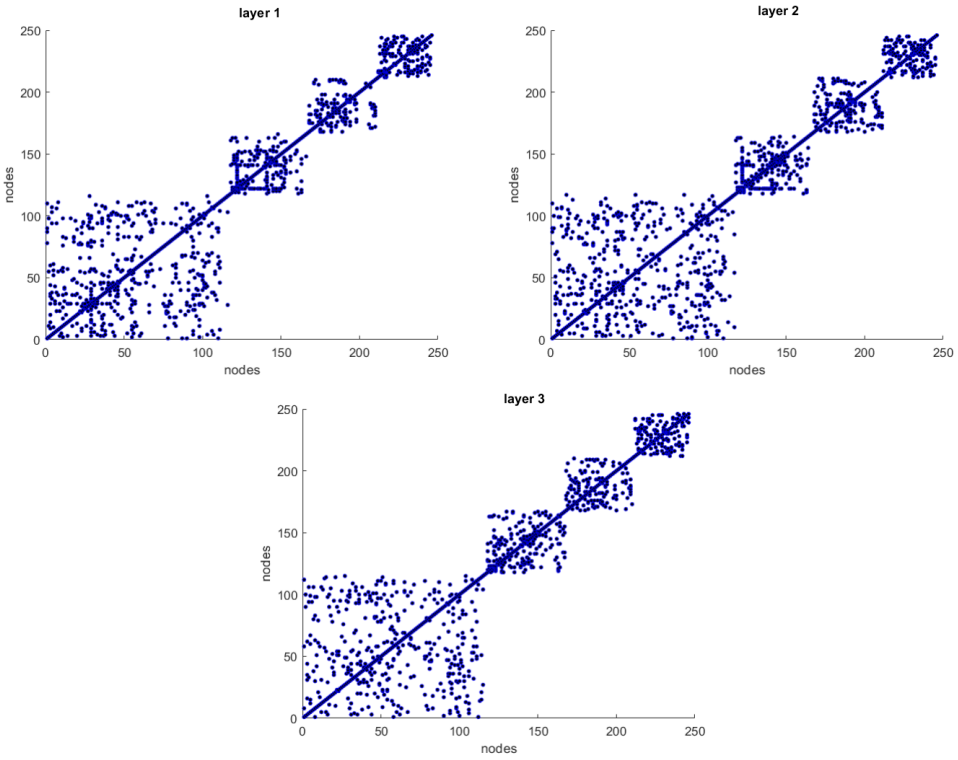


Figure 3.2: The layers of CKM.

3.2 Evaluation metrics

3.2.0.1 Normalized mutual information

The true community labels are known for all networks involved in the performance measurement process. Therefore, we chose Normalized mutual information (NMI) as the evaluation metric to evaluate the performance of the discussed community detection methods. NMI quantifies the information that two variables A and B share. In the case of community detection, A and B may correspond to two partitions of a given network, where A is the partition constructed by a community detection algorithm and B is the true underlying community structure. The NMI between two partitions A and B is defined in [40] as:

$$I_{AB} = -2 \cdot \frac{\sum_{i=1}^{N_M^A} \sum_{j=1}^{N_M^B} \log\left(\frac{n_{ij}^{AB} N}{n_i^A n_j^B}\right)}{\sum_{i=1}^{N_M^A} n_i^A \log(\frac{n_i^A}{N}) + \sum_{j=1}^{N_M^B} n_j^B \log(\frac{n_i^B}{N})}, \tag{3.2}$$

where N is the number of nodes in the network, n_i^A is the number of nodes in community i in partition A, n_{ij}^{AB} is the number of nodes that appear in both community i in the partition A, and community j in partition b. If the partitions A and B are independent, then the constructed partition A does not give us any information about the true partition B, so their NMI is zero. In contrast, if the partitions are identical, then their NMI is one. For example, given a network consisting of four

nodes n_1, n_2, n_3, n_4 with an underlying community structure $B = \{n_1, n_2, n_3, n_4\}$ and a constructed partition $A = \{\{n_1\}, \{n_2\}, \{n_3\}, \{n_4\}\}$, then knowledge of A does not provide any information about B in terms of its community structure, so their NMI is zero.

3.3 Parameter tuning

As shown in Sections 2.4.2 and 2.4.3, cross-validation in conjunction with grid search serve as formidable method for parameter tuning. The performance of a model with a given set of parameters is evaluated using NMI to estimate the error function.

The partitioning process of cross-validation for a network-based dataset is non-trivial. There exist several plausible approaches for partitioning the data. For example, one could start by partitioning the nodes into subsets and treat their induced subgraphs as the datasets for cross-validation. However, these subsets may not inherit the same community structure as the original graph, since several edges are deleted during the construction of the induced subgraphs. This creates a faulty training set which may steer the algorithm to detect the wrong type of community structure. A better strategy is to instead construct subsets that consist of the same nodes as in the original network and generate connections between them by randomly sampling edges from the original network [41]. As the edges are chosen randomly, the subsets are likely to inherit the same community structure as the original network. However, a problem with this approach is that the subsets may be very sparse as they only include a fraction of the edges of the original network. Hence, the algorithm is once again trained on a dataset that has different properties (sparsity) than the original network. To solve this issue we propose a novel approach for data partitioning for synthetic networks that creates a partitioning that shares all the same properties of the original network. The idea is to generate several networks from the same SBM as the original network and utilize them for training and validation in order to learn the parameters that best capture the underlying community structure. As they are generated from the same SBM, the training set inherits the same properties including sparsity, average node degree, number of nodes, number of interconnections within communities and overlap.

In order to achieve a good estimation of the generalized error, we generate a training set consisting of five networks generated from the same mSBM as the original network.

3.4 Convergence of BP

Due to short loops in the networks, the BP algorithm sometimes fails to converge to the desired point. Therefore, to get an accurate representation of the performance of BP, the algorithm is executed a total of 20 times for each experiment, and the result is averaged over all trials.

4

Results

In this chapter we present the results for the discussed community detection methods on synthetic and real-world multiplex networks with different properties and characteristics. The choice of parameters α_1 and α_2 is essential for obtaining good performance for CLMC. Therefore, we open this chapter with Section 4.1 where the results for the parameter tuning process of CLMC are shown. The results for CLMC in the sections following Section 4.1 refers to the results obtained by the model with the parameters α_1 and α_2 that yielded the lowest generalized error. To continue, in Section 4.2 we study the effect of unknown parameters. In Sections 4.3-4.7, different network properties are examined and compared to the performance of the discussed community detection methods on networks generated from the default generative model. Lastly, in Section 4.8 the results for real-world networks, including C.Elegans and CKM, are shown.

4.1 Impact of parameters α_1 and α_2

For the CLMC algorithm, we tune the parameters α_1 and α_2 in order to improve the performance for a network generated by the default SBM according to the description in Section 3.3. The parameters α_1 and α_2 takes values from $(0, 1)$ with a step length of 0.05. Figure 4.1 highlights the importance of the parameter tuning process for CLMC. Poor performance is achieved if the parameters are not chosen carefully. In 4.1 we see that relatively stable performance is obtained for a multiplex network generated from the default generative model when $1 > \alpha_1 > 0.5$ and $1 > \alpha_2 > 0$.

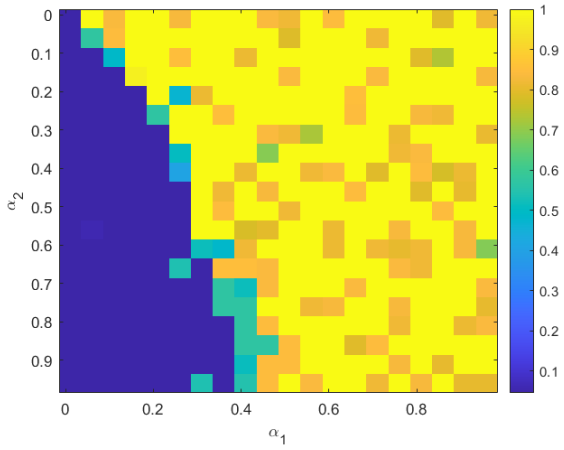


Figure 4.1: NMI for CLMC with different parameters $\alpha_1 \in [0, 1]$ and $\alpha_2 \in [0, 1]$ for a multiplex network generated by the default generative model.

4.2 Impact of unknown parameters

In this section the effect of unknown parameters is examined.

4.2.1 Impact of unknown number of communities

BP and CLMC both require an input parameter q that instructs the algorithm to choose from a set of q community labels when constructing a community structure. In many practical applications, the actual number of communities k is unknown. In this section, we study the effect of an unknown number of communities and whether it affects the performance of BP and CLMC.

For the flattening and layer by layer techniques for belief propagation, an unknown number of communities does not impact the result in any meaningful way. As long as q is equal or greater than the actual number of communities within the network, the BP algorithm converges, see Figure 4.2. In Figure 4.3, we see an example of how these algorithms converge to its solution. Regardless of having a pool of five communities to choose from, the algorithm tends to choose from one of two labels, leaving the remaining community labels unused. The labeling process of Multi-BP is much more uncertain. In Figure 4.4 we see that an unknown parameter q can significantly impact the performance of the global version of BP. The algorithm manages to label one of the communities correctly, however, the labels of the remaining nodes are chosen at random with equal probability from a pool of the four remaining community labels.

In Figure 4.5 we see that the extensions of CLMC achieve optimal performance when q is equal to the actual number of communities. The performance decreases as q diverge from the actual number of communities for both the flattening and layer by layer version. In Figure 4.6, we see an example output of CLMC for a multiplex network with $k = 2$ and $q = 5$. In this scenario, the algorithm manages to

partition one of the communities into perfect block structures but fails to identify the remaining block and instead partition it into four smaller blocks, steered by the faulty choice of $q = 5$.

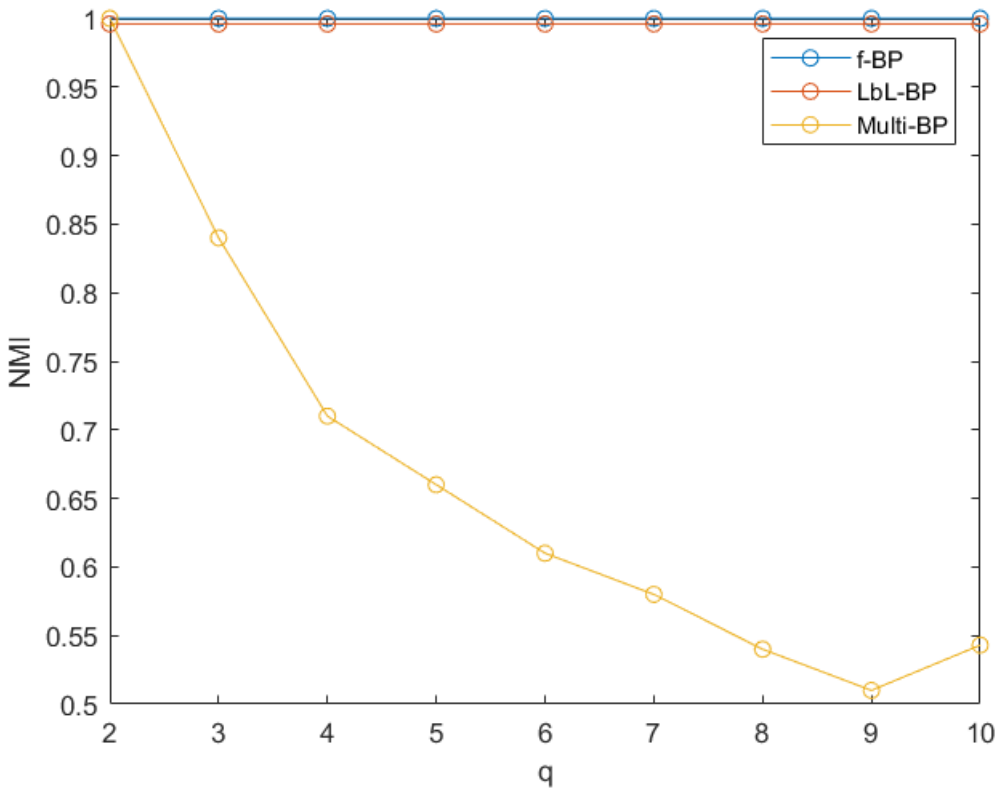


Figure 4.2: Performance of f-BP, LbL-BP, and Multi-BP for different values of q for a multiplex network generated from the default generative model. The results are averaged over 20 runs.

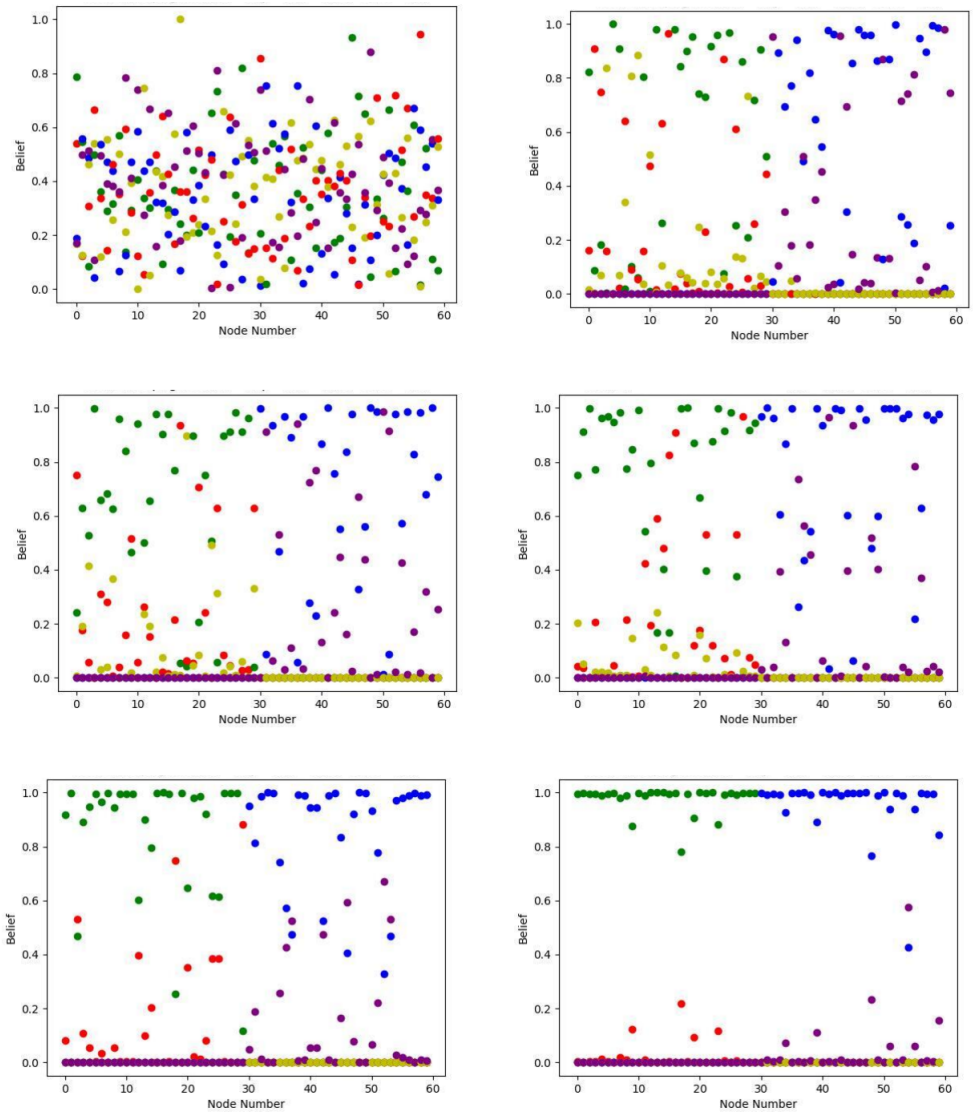


Figure 4.3: An example of an output generated by f-BP with $q = 5$ and $k = 2$ for a multiplex network generated by the default generative model. The beliefs for each community label are represented by different colors. The top-left image shows the initial beliefs for each node and the update process until convergence is shown in its chronological: top-left → top-right → middle-left → middle-right → bottom-left → bottom-right.

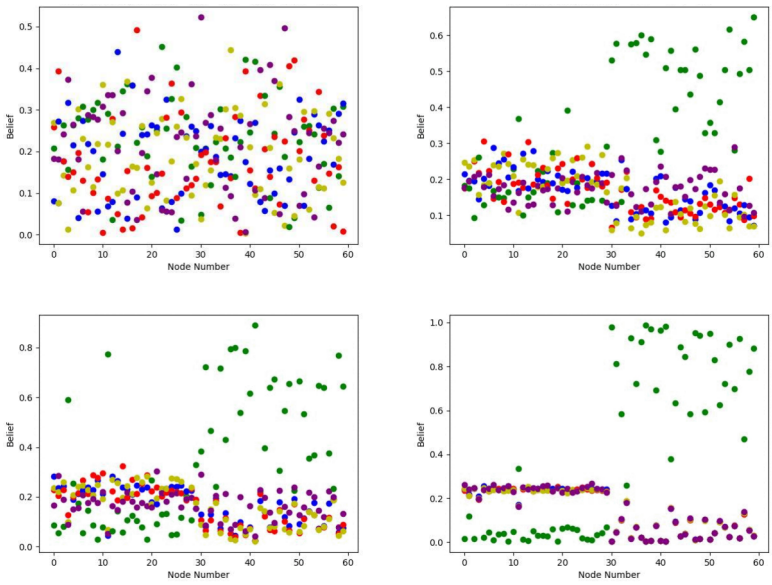


Figure 4.4: An example of an output generated by Multi-BP with $q = 5$ and $k = 2$ for a multiplex network generated by the default generative model. The beliefs for each community label are represented by different colors. The top-left image shows the initial beliefs for each node and the update process until convergence is shown in its chronological: top-left $\rightarrow$ top-right $\rightarrow$ bottom-left $\rightarrow$ bottom-right.

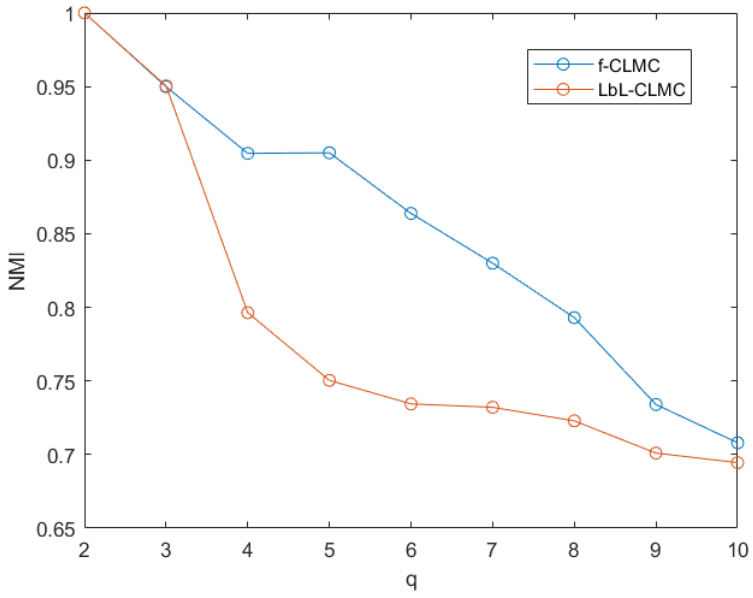


Figure 4.5: Performance of f-CLMC and LbL-CLMC, for different values of q for a multiplex network generated from the default generative model.

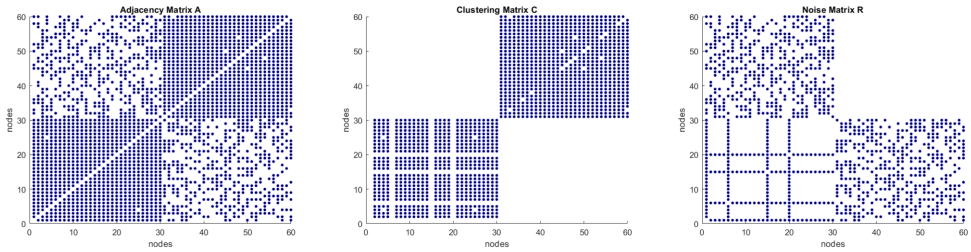


Figure 4.6: An example output of LbL-CLMC with $q = 5$ and $k = 2$ for one of the layers of a multiplex network generated by the default generative model.

4.2.1.1 Impact of a unknown block matrix C

In Figure 4.8 we see that all three versions of BP are relatively insensitive to the choice of $\hat{c}_{in}$, as long as $\hat{c}_{in}$ is greater than the true value c_{out}. When $\hat{c}_{in} < c_{out}$ the algorithm does not converge. Similarly, BP yields good performance as long as $\hat{c}_{out}$ is less than the true value of c_{in}. Due to the insensitivity of BP in regards to the model parameters $\hat{c}_{in}$ and $\hat{c}_{out}$, they are not required to be finely tuned during the training process.

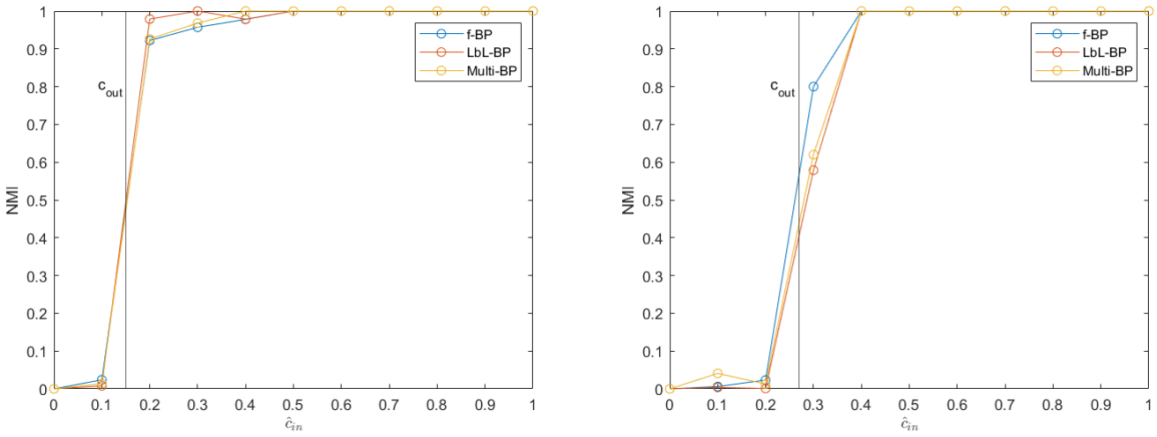


Figure 4.7: Performance of f-BP, LbL-BP and Multi-BP for different values of $\hat{c}_{in}$ for a two-layered multiplex network with $q(l) = k(l) = 2$, $N(l) = 60$, $N_1(l) = 30$, $N_2(l) = 30$, , $\epsilon(l) = 0.3$ and a true value $c_{in}(l) = 0.3$ (left) and $c_{in}(l) = 0.9$ (right) for $l = 1, 2$. The results are averaged over 20 runs.

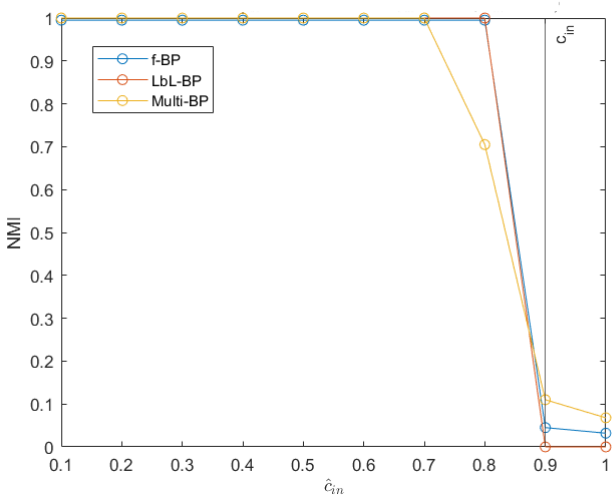


Figure 4.8: Performance of f-BP, LbL-BP and Multi-BP for different values of $\hat{c}_{out}$ for a two-layered multiplex network with $q(l) = k(l) = 2$, $N(l) = 60$, $N_1(l) = 30$, $N_2(l) = 30$, , $\epsilon(l) = 0.3$ and a true value $c_{in}(l) = 0.9$ (left) for $l = 1, 2$. The results are averaged over 20 runs.

4.3 CLMC decompositions

Here we showcase some examples of outputs generated by CLMC for different synthetic networks. In Figure 4.9 we see a network consisting of $N = 60$ nodes with a community structure of two communities of equal size with $c_{in} = 0.8$ and $c_{out} = 0.2$. Impressively, the algorithm manages to construct a near perfect clustering matrix consisting of two fully connected blocks where the missing connections within the communities are filled in and the overlaps are allocated to the noise matrix. This example displays the importance of the supplement matrix, as it clearly makes the clustering process much more effective as the missing edges are restored. In the second example in Figure 4.9 we see how effective the algorithm is even for relatively noisy networks. In Figure 4.10 we see a network consisting of $N = 120$ nodes with a community structure of four communities of equal sizes with $c_{in} = 1.0$ and $c_{out} = 0.5$. The algorithm manages to produce a near perfect clustering matrix with relatively few exceptions where interconnections are instead included in the noise matrix.

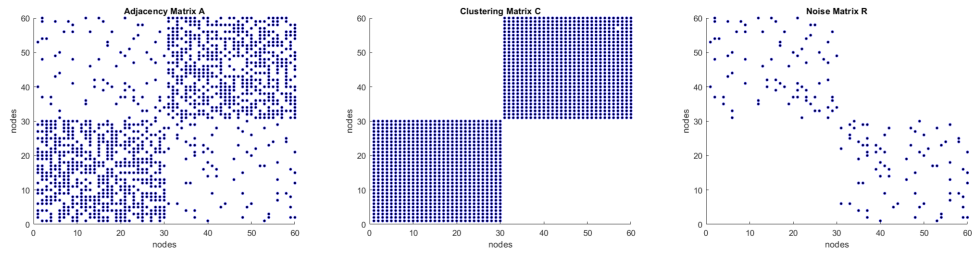


Figure 4.9: Output of LbL-CLMC for one of the layers for a multiplex network with $q = k = 2$, $c_{in} = 0.8$, $c_{out} = 0.2$, $N = 60$ and $N_1 = N_2 = 30$.

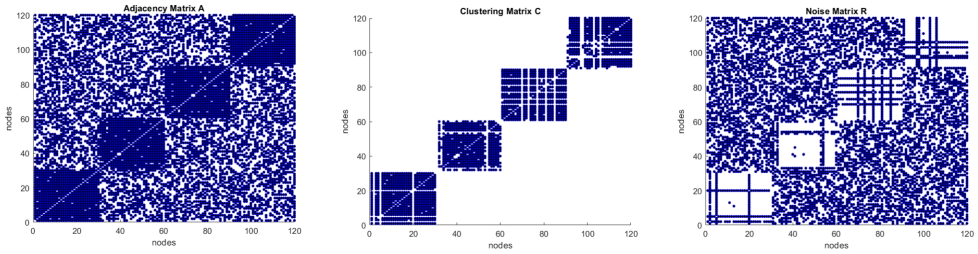


Figure 4.10: An example output generated by LbL-CLMC on one of the layers of a multiplex network with $q = k = 4$, $c_{in} = 1$, $c_{out} = 0.5$, $N = 120$ and $N_1 = N_2 = N_3 = N_4 = 30$.

4.4 Community sizes and number of communities

In this section we examine how the number of communities and their sizes affect the performance of the proposed community detection methods. As it is impossible to change the number of communities generated by the default mSBM without altering

the community sizes, we examine their properties simultaneously. In the Figures 4.11 and 4.12 we see the performance of the BP and CLMC extension respectively for a different number of communities of equal size. Unsurprisingly, the detectability of communities decrease as their size decrease. Additionally, the community detection algorithms become more susceptible to noise as the community sizes decrease.

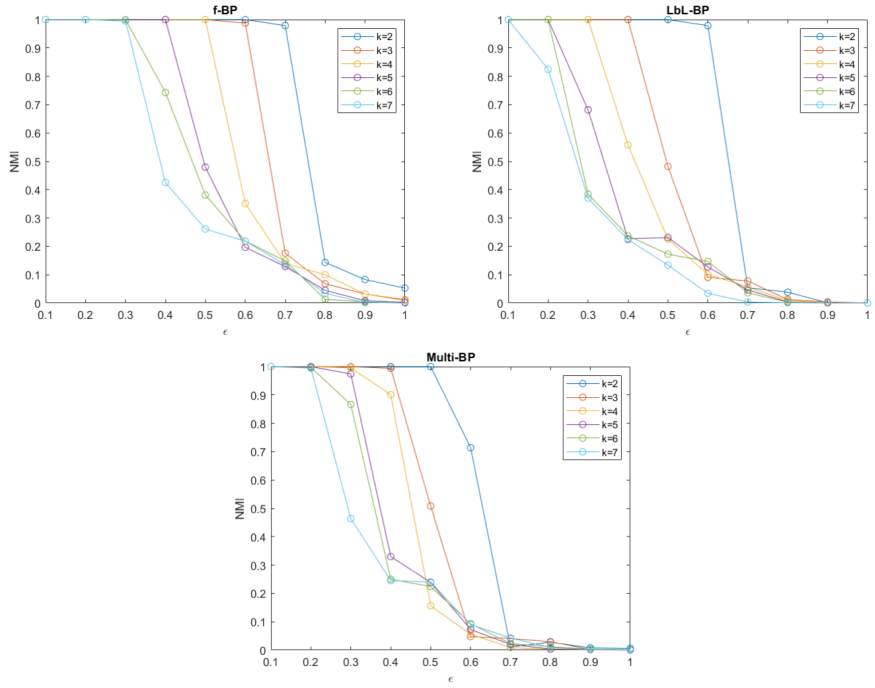


Figure 4.11: Performance of f-BP (top-left), LbL-BP (top-right) and Multi-BP (bottom) for a two-layered multiplex network with $N(l) = 60$, $c_{in}(l) = 0.9$, $c_{out}(l) = 0.3$ for $l = 1, 2$ for a varying number of communities of equal size.

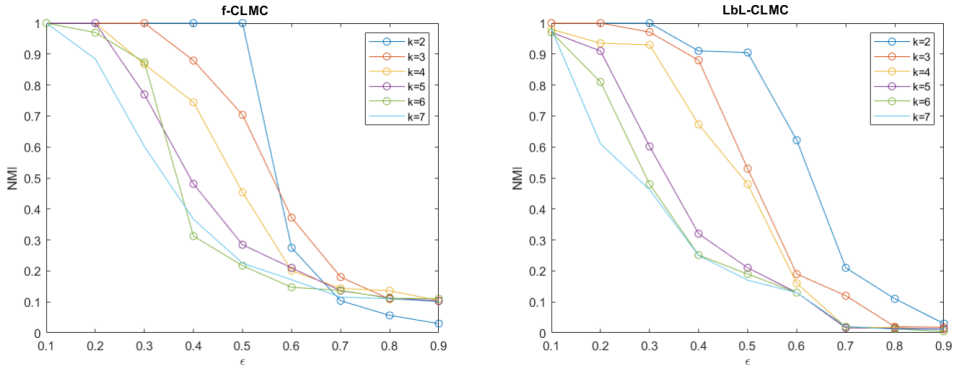


Figure 4.12: Performance of f-CLMC (left) and LbL-CLMC (right) for a two-layered multiplex network with $N(l) = 60$, $c_{in}(l) = 0.9$, $c_{out}(l) = 0.3$ for $l = 1, 2$ for a varying number of communities of equal size.

4.5 Sparsity

In this section, the impact of sparsity and the effectiveness of community detection techniques on sparse multiplex networks is examined. By varying the connectivity parameters c_{in} and c_{out} of the default generative model, whilst keeping the relation $\epsilon = c_{out}/c_{in} = 0.3$ fixed, the sparsity property is isolated. Figure 4.13 shows how sparsity affects the effectiveness of f-CLMC and LBL-CLMC for $epsilon = c_{out}/c_{in} = 0.3$. As the average node degree decreases, performance declines. This is most likely brought on by the growing number of missing edges that CLMC must fill in order to construct a low-rank matrix. A similar trend appears in the results for BP on sparse networks, where the performance once again declines as the average node degree decreases. For small average degrees, nodes can become isolated from their community which hinders them from receiving messages from other community members. Lastly, in Figures 4.13 and 4.14 we see that the flattening versions and Multi-BP yield better performance compared to the layer by layer techniques for sparse networks with a low average node degree. The added performance of the flattening technique on sparse networks is likely caused by the increased connectivity within communities achieved by its edge projection process.

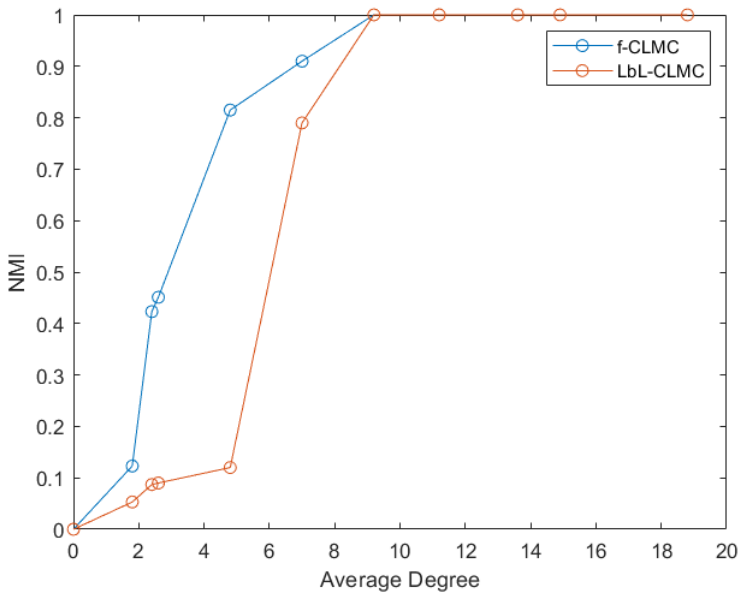


Figure 4.13: Performance of f-RMP and LbL-RMP for varying average degrees where the noise parameter $\epsilon = 0.3$ is kept fixed for a two-layered multiplex network with $k(l) = q(l) = 2$, $N(l) = 60$, $N_1(l) = N_2(l) = 30$ for $l = 1, 2$.

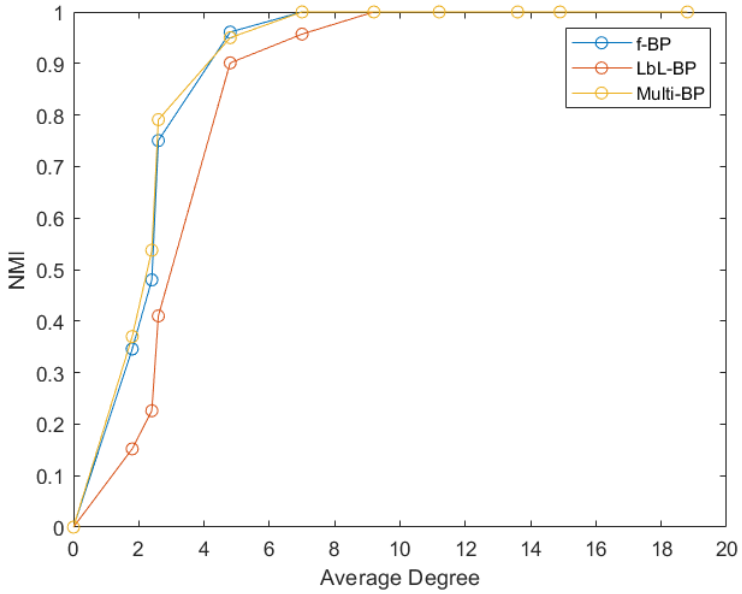


Figure 4.14: Performance of f-BP, LbL-BP and Multi-BP for varying average degrees where the noise parameter $\epsilon = 0.3$ is kept fixed for a two-layered multiplex network with $k(l) = q(l) = 2$, $N(l) = 60$, $N_1(l) = N_2(l) = 30$ for $l = 1, 2$.

4.6 Number of layers

In this section we show how the number of layers affect the performance of the discussed extension techniques.

4.6.1 Flattening

In Figure 2.6 we see the effectiveness of the flattening technique for detecting relatively dense interconnected community structures, even for high noise environments ($\epsilon = 0.5$). As the noise level increases, the span of optimal threshold values narrows. In high noise environments, noisy edges are present in more layers, requiring the filtering process to be more strict. Hence, performance increases as the threshold constant gets closer to the number of layers. This is true for densely interconnected community structures, however it is not obvious that good performance can be achieved for sparse networks, since increasing the threshold constant would imply that a large number of non-noisy edges are deleted which could negatively affect performance. In Figure 4.16 we see that good performance is achieved for medium sparsity ($c_{in} = 0.3$). However, for sparse networks ($c_{in} = 0.1$) good performance is only achievable for a low noise environment. In the case of high noise environment a large number of layers is required for flattening to achieve good performance for sparse networks.

Similar results are obtained for f-BP. Figure 4.18, 4.19 and 4.20 show the performance of f-BP for different thresholds for high, medium and low sparsity networks

respectively. We see that the flattening technique enhances the detectability of single-actor communities by removing noisy edges in densely connected networks whilst increasing the number of intra-connectivity in sparse networks.

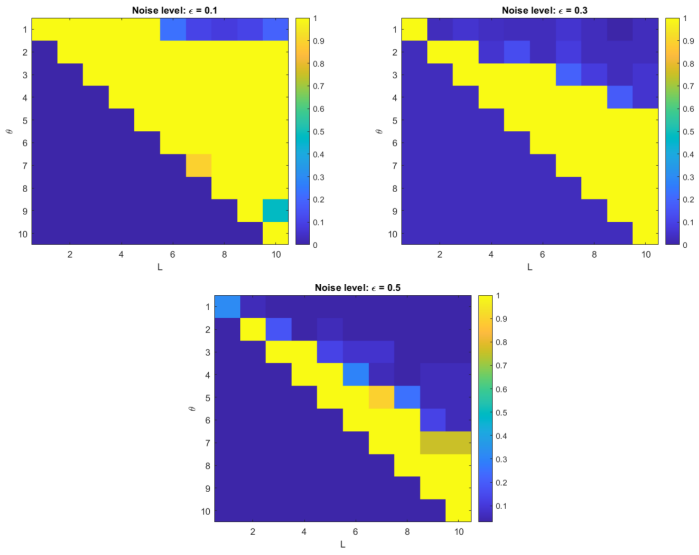


Figure 4.15: Performance of f-CLMC for different number of layers and values of θ for a multiplex network with $N(l) = 60$, $N_1(l) = 30$, $N_2(l) = 30$ and $c_{in}(l) = 0.9$ for $1 \leq l \leq L$ with noise levels $\epsilon = 0.1$ (upper-left), $\epsilon = 0.3$ (upper-right) and $\epsilon = 0.5$ (bottom).

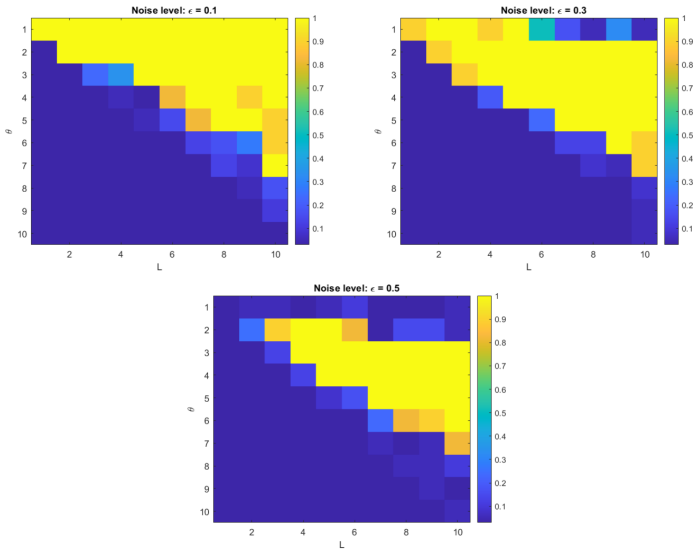


Figure 4.16: Performance of f-CLMC for different number of layer and values of θ for a multiplex network with $N(l) = 60$, $N_1(l) = 30$, $N_2(l) = 30$ and $c_{in}(l) = 0.3$ for $1 \leq l \leq L$ with noise levels $\epsilon = 0.1$ (upper-left), $\epsilon = 0.3$ (upper-right) and $\epsilon = 0.5$ (bottom).

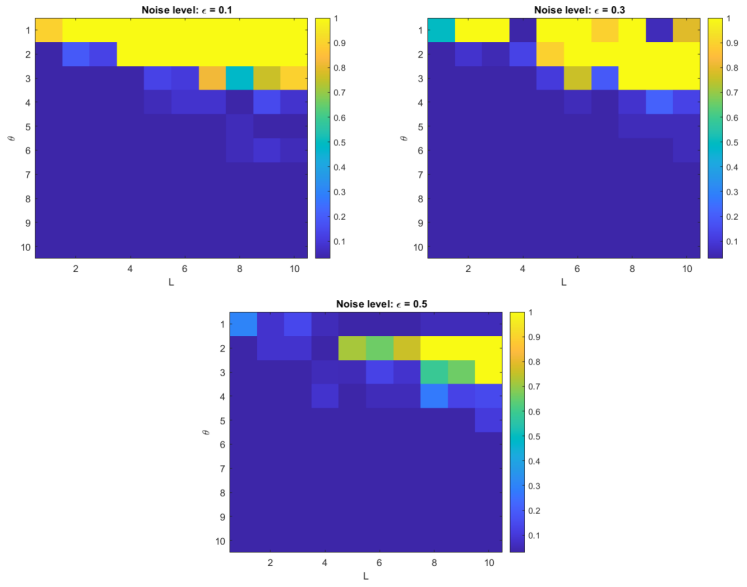


Figure 4.17: Performance of f-CLMC for different number of layers and values of θ for a multiplex network with $N(l) = 60$, $N_1(l) = 30$, $N_2(l) = 30$ and $c_{in}(l) = 0.1$ for $1 \leq l \leq L$ with noise levels $\epsilon = 0.1$ (upper-left), $\epsilon = 0.3$ (upper-right) and $\epsilon = 0.5$ (bottom).

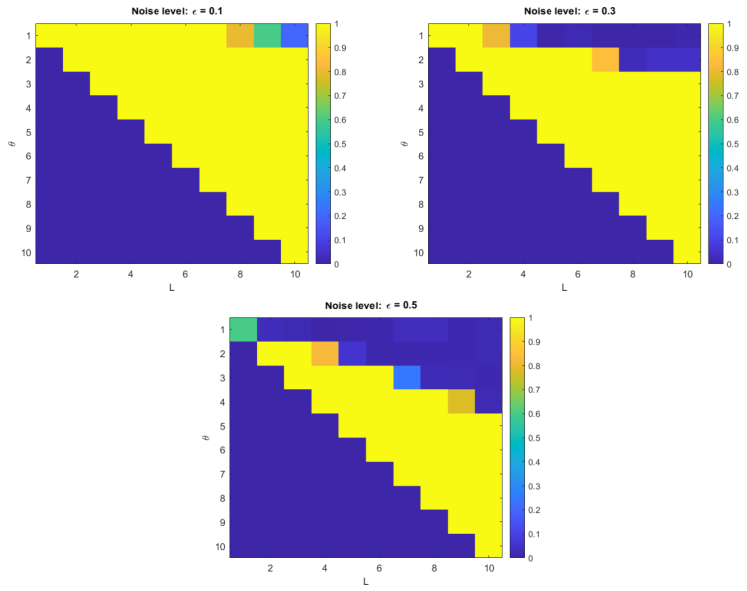


Figure 4.18: Performance of f-BP for different number of layers and values of θ for a multiplex network with $N(l) = 60$, $N_1(l) = 30$, $N_2(l) = 30$ and $c_{in}(l) = 0.9$ for $1 \leq l \leq L$ with noise levels $\epsilon = 0.1$ (upper-left), $\epsilon = 0.3$ (upper-right) and $\epsilon = 0.5$ (bottom). The results are averaged over 20 runs.

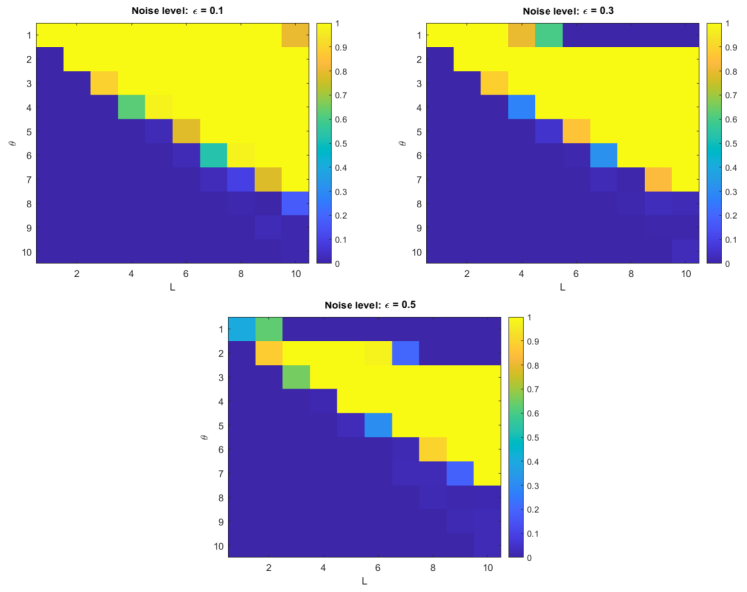


Figure 4.19: Performance of f-BP for different number of layers and values of θ for a multiplex network with $N(l) = 60$, $N_1(l) = 30$, $N_2(l) = 30$ and $c_{in}(l) = 0.3$ for $1 \leq l \leq L$ with noise levels $\epsilon = 0.1$ (upper-left), $\epsilon = 0.3$ (upper-right) and $\epsilon = 0.5$ (bottom). The results are averaged over 20 runs.

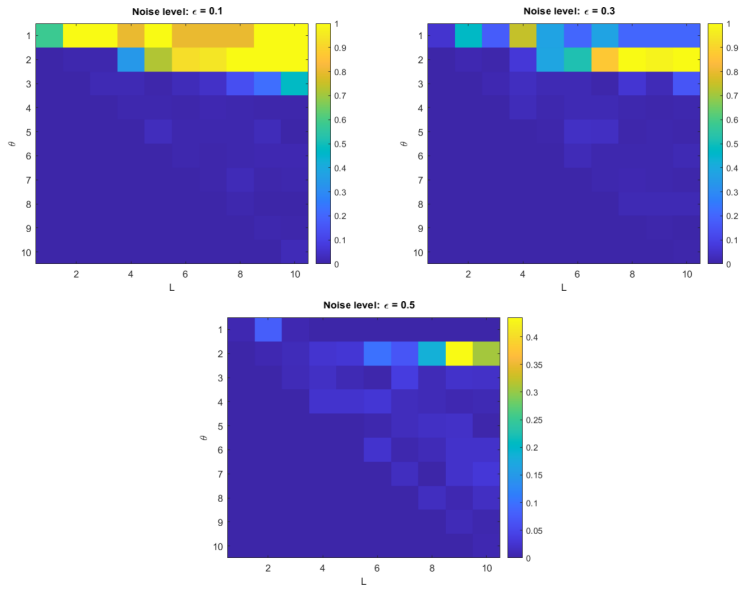


Figure 4.20: Performance of f-BP for different number of layers and values of θ for a multiplex network with $N(l) = 60$, $N_1(l) = 30$, $N_2(l) = 30$ and $c_{in}(l) = 0.1$ for $1 \leq l \leq L$ with noise levels $\epsilon = 0.1$ (upper-left), $\epsilon = 0.3$ (upper-right) and $\epsilon = 0.5$ (bottom). The results are averaged over 20 runs.

4.6.2 Layer-by-layer

In Figures 4.21, 4.22 and 4.23 we see the performance of LbL-CLMC for different threshold values on multiplex networks with a varying number of layers. The benefit of the layer by layer technique becomes more prevalent in high noise environments. When the noise is high, CLMC performs poorly on single-layered networks. However, by putting together a consensus from the obtained solutions from each layer, the layer-by-layer technique manages to improve performance and, unsurprisingly, an increased number of layers amplify the improvement. There is a limitation of the layer-by-layer technique compared to the flattening technique demonstrated by Figure 4.23. As shown in the previous section, the flattening techniques manage to improve performance on sparse networks by increasing the intraconnectivity within communities. However, for sparse networks the single-layered CLMC algorithm generates an output that approximately resembles a random assignment of community labels rather than the underlying community structure. Since the layer-by-layer techniques rely on the effectiveness of the corresponding single-layered algorithm, it does not significantly improve performance when the sparsity is too low or the noise is too high. In Figures 4.24, 4.25 and 4.26 we see that similar results are obtained for LbL-BP.

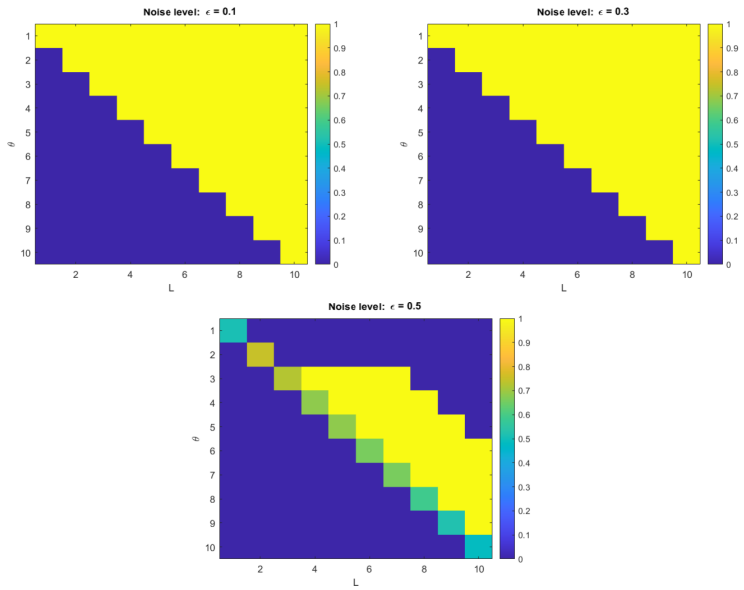


Figure 4.21: Performance of LbL-CLMC for different number of layers and values of θ for a multiplex network with $N(l) = 60$, $N_1(l) = 30$, $N_2(l) = 30$ and $c_{in}(l) = 0.9$ for $1 \leq l \leq L$ with noise levels $\epsilon = 0.1$ (upper-left), $\epsilon = 0.3$ (upper-right) and $\epsilon = 0.5$ (bottom).

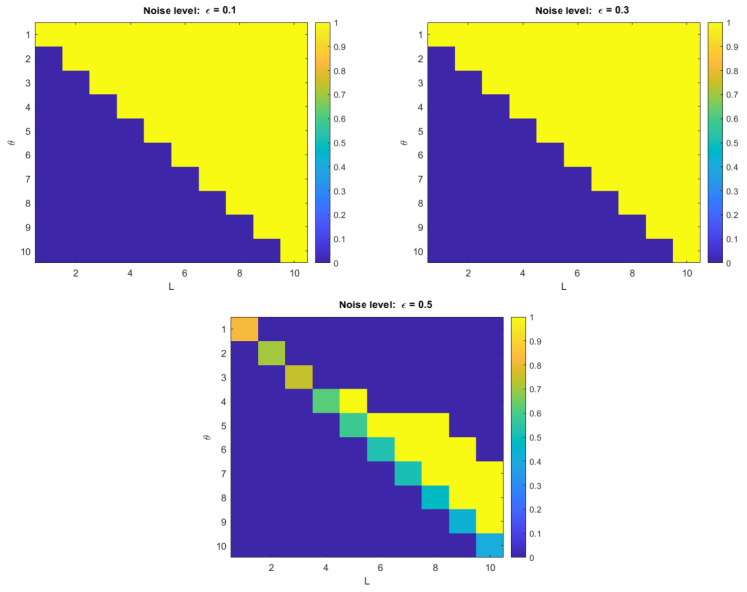


Figure 4.22: Performance of LbL-CLMC for different number of layers and values of θ for a multiplex network with $N(l) = 60$, $N_1(l) = 30$, $N_2(l) = 30$ and $c_{in}(l) = 0.3$ for $1 \leq l \leq L$ with noise levels $\epsilon = 0.1$ (upper-left), $\epsilon = 0.3$ (upper-right) and $\epsilon = 0.5$ (bottom)..

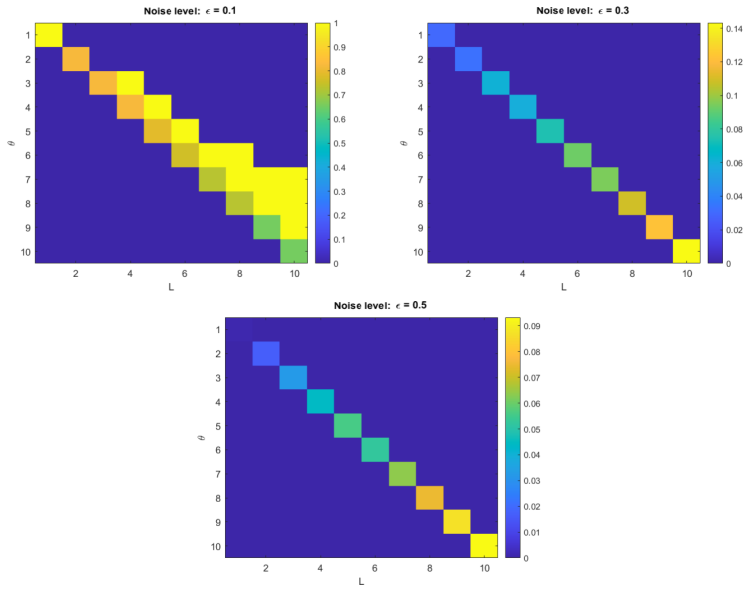


Figure 4.23: Performance of LbL-CLMC for different number of layers and values of θ for a multiplex network with $N(l) = 60$, $N_1(l) = 30$, $N_2(l) = 30$ and $c_{in}(l) = 0.1$ for $1 \leq l \leq L$ with noise levels $\epsilon = 0.1$ (upper-left), $\epsilon = 0.3$ (upper-right) and $\epsilon = 0.5$ (bottom).

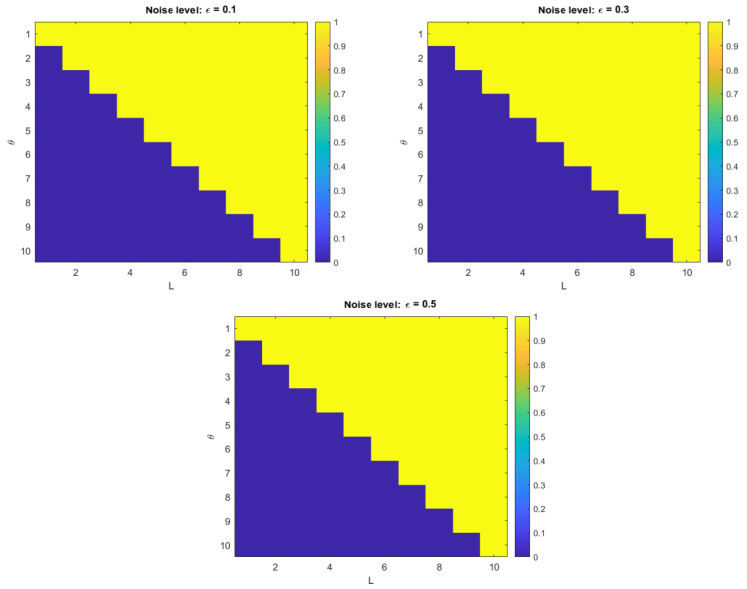


Figure 4.24: Performance of LbL-BP for different number of layers and values of θ for a multiplex network with $N(l) = 60$, $N_1(l) = 30$, $N_2(l) = 30$ and $c_{in}(l) = 0.9$ for $1 \leq l \leq L$ with noise levels $\epsilon = 0.1$ (upper-left), $\epsilon = 0.3$ (upper-right) and $\epsilon = 0.5$ (bottom). The results are averaged over 20 runs.

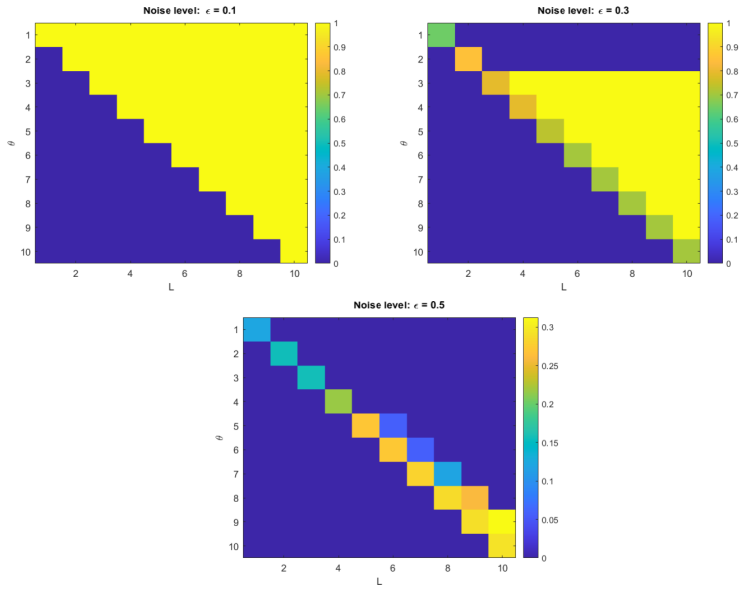


Figure 4.25: Performance of LbL-BP for different number of layers and values of θ for a multiplex network with $N(l) = 60$, $N_1(l) = 30$, $N_2(l) = 30$ and $c_{in}(l) = 0.3$ for $1 \leq l \leq L$ with noise levels $\epsilon = 0.1$ (upper-left), $\epsilon = 0.3$ (upper-right) and $\epsilon = 0.5$ (bottom). The results are averaged over 20 runs.

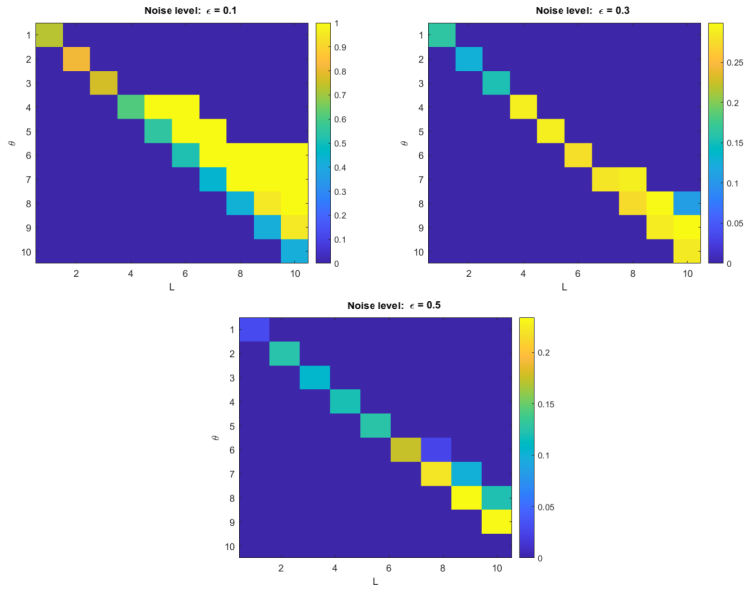


Figure 4.26: Performance of LbL-BP for different number of layers and values of θ for a multiplex network with $N(l) = 60$, $N_1(l) = 30$, $N_2(l) = 30$ and $c_{in}(l) = 0.1$ for $1 \leq l \leq L$ with noise levels $\epsilon = 0.1$ (upper-left), $\epsilon = 0.3$ (upper-right) and $\epsilon = 0.5$ (bottom). The results are averaged over 20 runs.

4.6.3 Multi-BP

In Figure 4.27 we see the performance of Multi-BP on multiplex networks with a varying number of layers. The performance decreases as the number of layers increase. Much like a neural network, as the number of layers increases, the belief propagation algorithm starts to resemble a block-box as its underlying machinery becomes more complex. Hence, it is difficult to pinpoint the reason for the drop in performance. A plausible explanation is that the introduction of the factor nodes enforcing the WPP increases the number of short loops which causes the beliefs to converge to undesirable points [8].

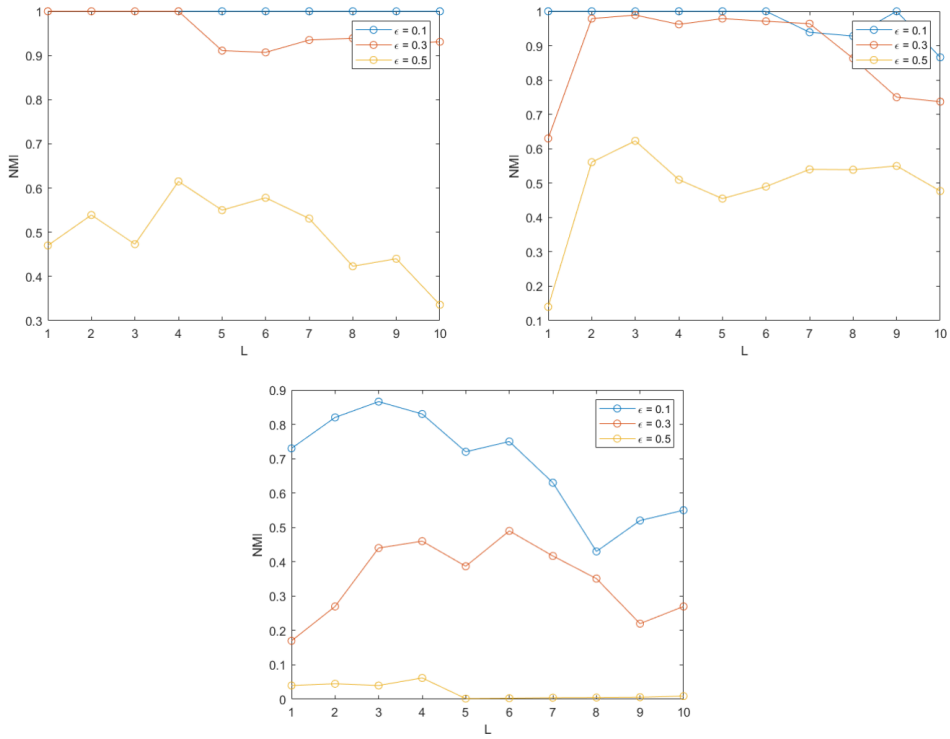


Figure 4.27: Performance of Multi-BP for different number of layers for a multiplex network with $N(l) = 60$, $N_1(l) = 30$, $N_2(l) = 30$ and $c_{in}(l) = 0.9$ (upper-left), $c_{in}(l) = 0.3$ (upper-right) and $c_{in}(l) = 0.1$ (bottom) for $1 \leq l \leq L$ for different noise levels $\epsilon = 0.1, 0.3, 0.5$. The results are averaged over 20 runs.

4.7 Complex community structures

In Figure 4.28 we see the ability of Multi-BP to detect complex community structures. The results suggest that the introduction of the WPP factor node enable Multi-BP to fuse together the underlying complex community structure, at least in low noise environments ($\epsilon \leq 0.3$).

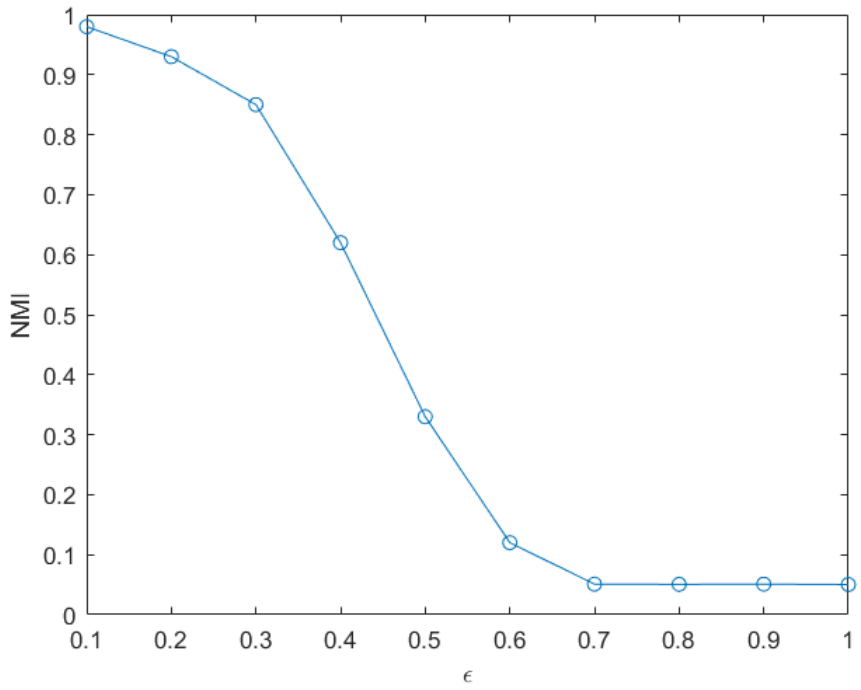


Figure 4.28: Performance of Multi-BP for a multiplex network with two layers with $N_1(l_1) = N_2(l_1) = 30$ and $N_1(l_2) = 30, N_2(l_2) = N_3(l_2) = 15$ for different noise levels.

4.8 Real-world data

In this section, the results for real-word networks for the discussed community detection methods are presented and compared to a benchmark provided by [33] which includes the performance of several other community detection methods: *Centroid-Coreg*, *PM*, *RMSC*, *SCML*, *CSNMF*, *CPNMF*, *CSNMTF*, *DiMMA*, *2CMV*, *Mx-CRTSA* for C.ELEGANS and CKM. The comparison results for C.ElEGANS and CKM is presented in the tables 4.1 and 4.2 respectively. In Figures 4.29, 4.30, 4.31 and 4.32 the performance for different parameter values α_1 and α_2 is presented for f-CLMC and LbL-CLMC. The performance of the proposed extensions of CLMC and BP are in line with the other community detection methods. The flattened version of CLMC performed particularly well, obtaining the best performance with a score of 1.0 for the CKM network. The community structure produced by Multi-BP for CKM was inferior compared to the other discussed community detection meth-

Table 4.1: Performance comparison between different community detection methods for the C.ELEGANS multiplex network.

C.ELEGANS	
Method	NMI
f-BP	0.470
LbL-BP	0.432
Multi-BP	0.448
f-CLMC	0.452
LbL-CLMC	0.442
CentroidCoreg	0.432
PM	0.427
RMSC	0.303
SCML	0.448
CSNMF	0.442
CPNMF	0.382
CSNMTF	0.461
DiMMA	0.341
2CMV	0.334
Mx-CRTSA	0.509

ods, most likely due to the added number of short loops introduced by the added inter-layered factor nodes.

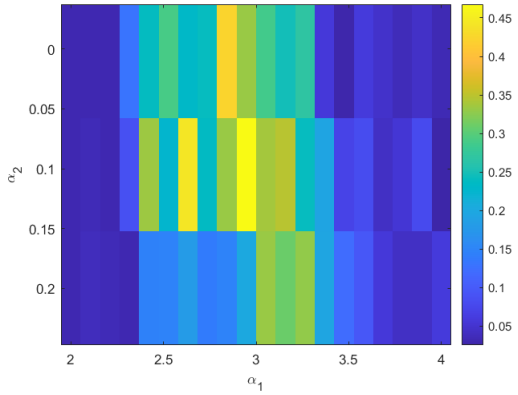


Figure 4.29: Performance of f-CLMC for the flattened C.ELEGANS network for different parameter values α_1 and α_2.

Table 4.2: Performance comparison between different community detection methods for the CKM multiplex network.

CKM	
Method	NMI
f-BP	0.853
LbL-BP	0.899
Multi-BP	0.631
f-CLMC	1.0
LbL-CLMC	0.989
CentroidCoreg	0.948
PM	0.032
RMSC	0.821
SCML	0.958
CSNMF	0.821
CPNMF	0.017
CSNMTF	0.963
DiMMA	0.472
2CMV	0.778
Mx-CRTSA	1.0

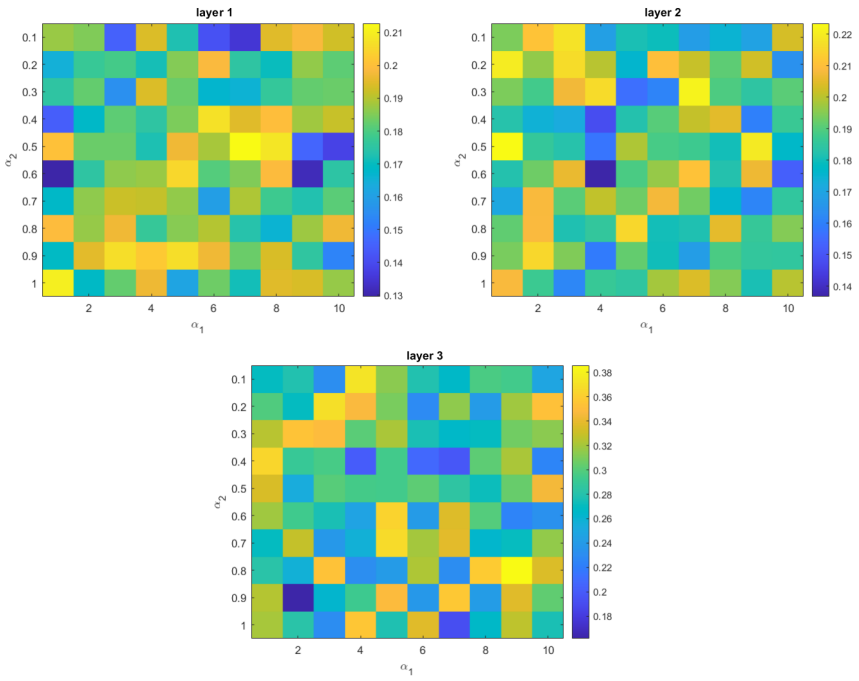


Figure 4.30: Performance of LbL-CLMC for the layers of C.ELEGANS for different parameter values α_1 and α_2.

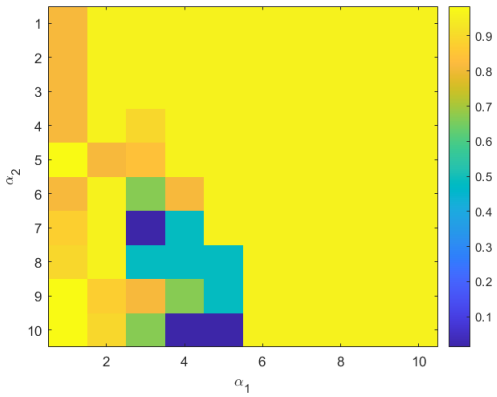


Figure 4.31: Performance of f-CLMC for flattened CKM network for different parameter values α_1 and α_2.

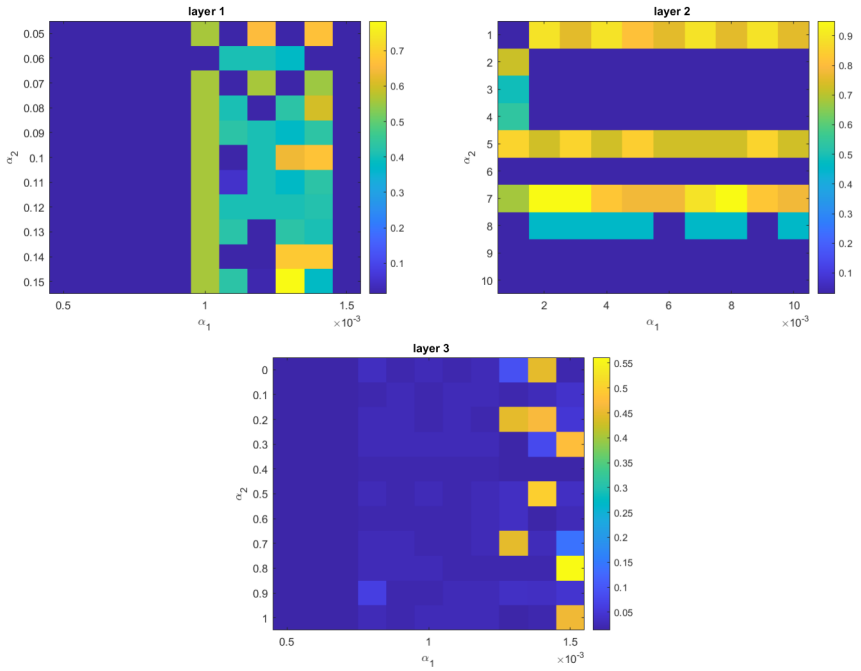


Figure 4.32: Performance of LbL-CLMC for the layers of CKM for different parameter values α_1 and α_2.

5

Conclusion

In this paper, several multiplex extensions of the community detection methods, belief propagation and cluster-driven low-rank matrix completion are proposed. For both algorithms, a flattening and a layer by layer extension technique is implemented. Additionally, a global algorithm for BP that operates directly on the multiplex network is discussed. The performance of the proposed extension techniques is assessed on various networks, both synthetic and real-world multiplex networks. The flattening and layer-by-layer versions exhibit an enhanced ability to detect single-actor community structures in multiplex networks due to an improved robustness to noise and sparsity. This effect is even more prevalent as the number of available layers increase. The Multi-BP algorithm shows good performance and high robustness to noise and manages to detect complex community structures, in addition to single-actor structures. However, an increased number of layers do not seem to improve performance of Multi-BP. The range of optimal model parameters for the extension techniques of BP appear to be relatively large and obtain good performance as long as the connectivity parameters satisfy $c_{in} > c_{out}$. However, the Multi-BP version shows increased sensitivity to an unknown number of communities. The performance of the extensions of CLMC is sensitive to the choice of parameters α_1 and α_2. However, the proposed parameter tuning technique is effective for finding the optimal model parameters.

Bibliography

[1] S. Fortunato, "Community detection in graphs," *Physics Reports*, vol. 486, no. 3, pp. 75–174, 2010, ISSN: 0370-1573. DOI: https://doi.org/10.1016/j.physrep.2009.11.002. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0370157309002841.

[2] M. Magnani, L. Rossi, and D. Vega, "Analysis of multiplex social networks with r," *Journal of Statistical Software*, vol. 98, no. 8, pp. 1–30, 2021. DOI: 10.18637/jss.v098.i08. [Online]. Available: https://www.jstatsoft.org/index.php/jss/article/view/v098i08.

[3] L. M. Sanchez-Rodriguez, Y. Iturria-Medina, P. Mouches, and R. C. Sotero, "Detecting brain network communities: Considering the role of information flow and its different temporal scales," *NeuroImage*, vol. 225, p. 117 431, 2021, ISSN: 1053-8119. DOI: https://doi.org/10.1016/j.neuroimage.2020.117431. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1053811920309162.

[4] N. Verstraete, G. Jurman, G. Bertagnolli, A. Ghavasieh, V. Pancaldi, and M. De Domenico, "Covmulnet19, integrating proteins, diseases, drugs, and symptoms: A network medicine approach to covid-19," *Network and Systems Medicine*, vol. 3, pp. 130–141, Nov. 2020. DOI: 10.1089/nsm.2020.0011.

[5] M. Magnani, O. Hanteer, R. Interdonato, L. Rossi, and A. Tagarelli, *Community detection in multiplex networks*, 2021. arXiv: 1910.07646 [cs.SI].

[6] A. Decelle, F. Krzakala, C. Moore, and L. Zdeborová, "Asymptotic analysis of the stochastic block model for modular networks and its algorithmic applications," *Physical Review E*, vol. 84, no. 6, Dec. 2011. DOI: 10.1103/physreve.84.066106. [Online]. Available: https://doi.org/10.1103%2Fphysreve.84.066106.

[7] J. Shao, Z. Zhang, Z. Yu, J. Wang, Y. Zhao, and Q. Yang, "Community detection and link prediction via cluster-driven low-rank matrix completion," in *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI-19*, International Joint Conferences on Artificial Intelligence Organization, Jul. 2019, pp. 3382–3388. DOI: 10.24963/ijcai.2019/469. [Online]. Available: https://doi.org/10.24963/ijcai.2019/469.

[8] Y. Huang, A. Panahi, H. Krim, and L. Dai, "Community detection and improved detectability in multiplex networks," *IEEE Transactions on Network Science and Engineering*, vol. 7, no. 3, pp. 1697–1709, 2020. DOI: 10.1109/TNSE.2019.2949036.

[9] J. Reichardt and M. Leone, "(un)detectable cluster structure in sparse networks," *Physical Review Letters*, vol. 101, no. 7, Aug. 2008. DOI: 10.1103/physrevlett.101.078701. [Online]. Available: https://doi.org/10.1103%2Fphysrevlett.101.078701.

[10] A. Decelle, F. Krzakala, C. Moore, and L. Zdeborová, "Inference and phase transitions in the detection of modules in sparse networks," *Physical Review Letters*, vol. 107, no. 6, Aug. 2011. DOI: 10.1103/physrevlett.107.065701. [Online]. Available: https://doi.org/10.1103%2Fphysrevlett.107.065701.

[11] T. Pontoizeau, "Community detection : Computational complexity and approximation," Ph.D. dissertation, Jun. 2018.

[12] C. Lee and D. J. Wilkinson, "A review of stochastic block models and extensions for graph clustering," *Applied Network Science*, vol. 4, no. 1, Dec. 2019. DOI: 10.1007/s41109-019-0232-2. [Online]. Available: https://doi.org/10.1007%2Fs41109-019-0232-2.

[13] C. Lee and D. J. Wilkinson, "A review of stochastic block models and extensions for graph clustering," *Applied Network Science*, vol. 4, no. 1, Dec. 2019. DOI: 10.1007/s41109-019-0232-2. [Online]. Available: https://doi.org/10.1007%2Fs41109-019-0232-2.

[14] C. Lee and D. J. Wilkinson, "A review of stochastic block models and extensions for graph clustering," *Applied Network Science*, vol. 4, no. 1, Dec. 2019. DOI: 10.1007/s41109-019-0232-2. [Online]. Available: https://doi.org/10.1007%2Fs41109-019-0232-2.

[15] E. Abbe, *Community detection and stochastic block models*, 2022. arXiv: 1703.10146 [math.PR].

[16] M. Mézard and A. Montanari, "Information, physics, and computation," 2009. [Online]. Available: https://web.stanford.edu/~montanar/RESEARCH/book.html.

[17] D. Koller and N. Friedman, *Probabilistic Graphical Models: Principles and Techniques* (Adaptive computation and machine learning). MIT Press, 2009, ISBN: 9780262013192. [Online]. Available: https://books.google.co.in/books?id=7dzpHCHzNQ4C.

[18] C. Knoll, *Understanding the behavior of belief propagation*, 2022. arXiv: 2209.05464 [cs.AI].

[19] B. Ames and S. Vavasis, *Nuclear norm minimization for the planted clique and biclique problems*, 2009. arXiv: 0901.3348 [cs.DS].

[20] A. Blumer, A. Ehrenfeucht, D. Haussler, and M. K. Warmuth, "Occam's razor," *Information Processing Letters*, vol. 24, no. 6, pp. 377–380, 1987, ISSN: 0020-0190. DOI: https://doi.org/10.1016/0020-0190(87)90114-1. [Online]. Available: https://www.sciencedirect.com/science/article/pii/0020019087901141.

[21] M. Fazel, H. Hindi, and S. Boyd, "Rank minimization and applications in system theory," in *Proceedings of the 2004 American Control Conference*, vol. 4, 2004, 3273–3278 vol.4. DOI: 10.23919/ACC.2004.1384521.

[22] R. Cabral, F. De la Torre, J. P. Costeira, and A. Bernardino, "Unifying nuclear norm and bilinear factorization approaches for low-rank matrix decom-

position," in *2013 IEEE International Conference on Computer Vision*, 2013, pp. 2488–2495. DOI: 10.1109/ICCV.2013.309.

[23] K. Fan, "On a theorem of weyl concerning eigenvalues of linear transformations i*," *Proceedings of the National Academy of Sciences*, vol. 35, no. 11, pp. 652–655, 1949. DOI: 10.1073/pnas.35.11.652. eprint: https://www.pnas.org/doi/pdf/10.1073/pnas.35.11.652. [Online]. Available: https://www.pnas.org/doi/abs/10.1073/pnas.35.11.652.

[24] S. Boyd, N. Parikh, E. Chu, B. Peleato, and J. Eckstein, "Distributed optimization and statistical learning via the alternating direction method of multipliers," *Foundations and Trendső in Machine Learning*, vol. 3, no. 1, pp. 1–122, 2011, ISSN: 1935-8237. DOI: 10.1561/2200000016. [Online]. Available: http://dx.doi.org/10.1561/2200000016.

[25] J.-F. Cai, E. J. Candes, and Z. Shen, *A singular value thresholding algorithm for matrix completion*, 2008. arXiv: 0810.3286 [math.OC].

[26] P. Favati, G. Lotti, O. Menchi, and F. Romani, "Construction of the similarity matrix for the spectral clustering method: Numerical experiments," *Journal of Computational and Applied Mathematics*, vol. 375, p. 112 795, 2020, ISSN: 0377-0427. DOI: https://doi.org/10.1016/j.cam.2020.112795. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0377042720300868.

[27] A. C. Kinsley, G. Rossi, M. J. Silk, and K. VanderWaal, "Multilayer and multiplex networks: An introduction to their use in veterinary epidemiology," *Frontiers in Veterinary Science*, vol. 7, 2020, ISSN: 2297-1769. DOI: 10.3389/fvets.2020.00596. [Online]. Available: https://www.frontiersin.org/articles/10.3389/fvets.2020.00596.

[28] P. Barbillon, S. Donnet, E. Lazega, and A. Bar-Hen, *Stochastic block models for multiplex networks: An application to networks of researchers*, 2015. arXiv: 1501.06444 [stat.ME].

[29] D. Taylor, S. Shai, N. Stanley, and P. J. Mucha, "Enhanced detectability of community structure in multilayer networks through layer aggregation," *Phys. Rev. Lett.*, vol. 116, p. 228 301, 22 Jun. 2016. DOI: 10.1103/PhysRevLett.116.228301. [Online]. Available: https://link.aps.org/doi/10.1103/PhysRevLett.116.228301.

[30] J. Kim, J.-G. Lee, and S. Lim, "Differential flattening: A novel framework for community detection in multi-layer graphs," *ACM Transactions on Intelligent Systems and Technology*, vol. 8, p. 27, Jan. 2017. DOI: 10.1145/2898362.

[31] D. Mandaglio, A. Amelio, and A. Tagarelli, *Consensus community detection in multilayer networks using parameter-free graph pruning*, 2018. arXiv: 1804.06653 [cs.DB].

[32] A. Tagarelli, A. Amelio, and F. Gullo, "Ensemble-based community detection in multilayer networks," *Data Mining and Knowledge Discovery*, vol. 31, Sep. 2017. DOI: 10.1007/s10618-017-0528-8.

[33] E. Al-sharoa, M. Al-wardat, M. Al-khassaweneh, and A. Al Bataineh, "Discovering community structure in multiplex networks via a co-regularized robust tensor-based spectral approach," *Applied Sciences*, vol. 13, no. 4, 2023, ISSN:

2076-3417. DOI: 10.3390/app13042514. [Online]. Available: https://www.mdpi.com/2076-3417/13/4/2514.

[34] J. Bergstra and Y. Bengio, "Random search for hyper-parameter optimization," *The Journal of Machine Learning Research*, vol. 13, pp. 281–305, Mar. 2012. [Online]. Available: http://www.jmlr.org/papers/volume13/bergstra12a/bergstra12a.pdf.

[35] B. Ghojogh and M. Crowley, *The theory behind overfitting, cross validation, regularization, bagging, and boosting: Tutorial*, May 2019. arXiv: 1905.12787 [stat.ML]. [Online]. Available: https://arxiv.org/abs/1905.12787.

[36] M. Markatou, H. Tian, S. Biswas, and G. Hripcsak, "Analysis of variance of cross-validation estimators of the generalization error.," *Journal of Machine Learning Research*, vol. 6, pp. 1127–1168, Jul. 2005. [Online]. Available: http://www.jmlr.org/papers/volume6/markatou05a/markatou05a.pdf.

[37] M. Claesen and B. De Moor, *Hyperparameter search in machine learning*, Feb. 2015. arXiv: 1502.02127 [cs.LG]. [Online]. Available: https://arxiv.org/abs/1502.02127.

[38] J. Coleman, E. Katz, and H. Menzel, "The diffusion of an innovation among physicians," *Sociometry*, vol. 20, no. 4, pp. 253–270, 1957, ISSN: 00380431. [Online]. Available: http://www.jstor.org/stable/2785979 (visited on 06/10/2023).

[39] B. L. Chen, D. H. Hall, and D. B. Chklovskii, "Wiring optimization can relate neuronal structure and function," *Proceedings of the National Academy of Sciences*, vol. 103, no. 12, pp. 4723–4728, 2006. DOI: 10.1073/pnas.0506806103. eprint: https://www.pnas.org/doi/pdf/10.1073/pnas.0506806103. [Online]. Available: https://www.pnas.org/doi/abs/10.1073/pnas.0506806103.

[40] R. Guimerà, M. Sales-Pardo, and L. Amaral, "Module identification in bipartite and directed networks," *Physical review. E, Statistical, nonlinear, and soft matter physics*, vol. 76, p. 036 102, Oct. 2007. DOI: 10.1103/PhysRevE.76.036102.

[41] T. Li, E. Levina, and J. Zhu, *Network cross-validation by edge sampling*, 2020. arXiv: 1612.04717 [stat.ME].