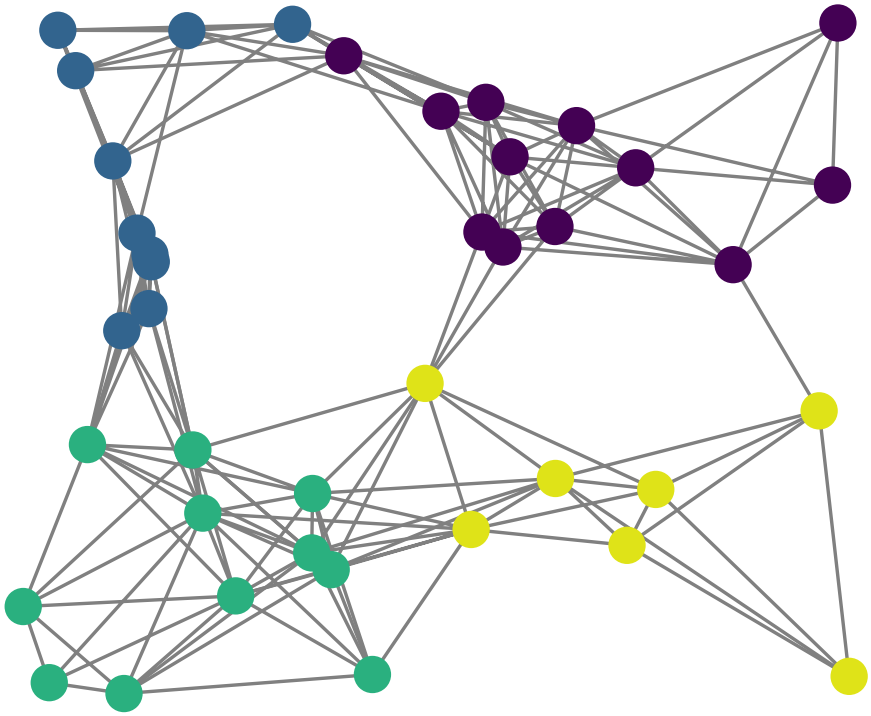
Circuits and Systems
Mekelweg 4,
2628 CD Delft
The Netherlands
https://cas.tudelft.nl/

CAS-2021-5037751

M.Sc. Thesis

Adaptive Graph Partition Methods for Structured Graphs

Yanbin He B.Sc.



Faculty of Electrical Engineering, Mathematics and Computer Science **Delft University of Technology**

Adaptive Graph Partition Methods for Structured Graphs

Thesis

submitted in partial fulfillment of the
requirements for the degree of

Master of Science

in

Electrical Engineering

by

Yanbin He B.Sc.
born in Xian Yang, China

This work was performed in:

Circuits and Systems Group
Department of Microelectronics
Faculty of Electrical Engineering, Mathematics and Computer Science
Delft University of Technology

Delft University of Technology

Copyright © 2021 Circuits and Systems Group
All rights reserved.

Delft University of Technology
Department of
Microelectronics

The undersigned hereby certify that they have read and recommend to the Faculty of Electrical Engineering, Mathematics and Computer Science for acceptance a thesis entitled **"Adaptive Graph Partition Methods for Structured Graphs"** by **Yanbin He B.Sc.** in partial fulfillment of the requirements for the degree of **Master of Science**.

Dated: October 22 2021

Chairman: _____
prof.dr.ir. G.J.T. Leus

Advisor: _____
dr. M.A. Coutino

Committee Members: _____
prof.dr. Huijuan Wang

_____

Abstract

Graphs can be models for many real-world systems, where nodes indicate the entities and edges indicate the pairwise connections in between. In various cases, it is important to detect informative subsets of nodes such that the nodes within the subsets are 'closer' to each other. For example, in a cellular network, determining appropriate node subsets can reduce the operation costs. A subset is usually called a *cluster*. This leads to the graph clustering problem. Furthermore, plenty of systems in the real world are changing over time, and consequently, graphs as models vary with time as well. It is thus also important to update the clusters when the graph changes.

In this thesis work, we studied two problems from the cellular network background. We needed to partition graphs that have certain structures and cluster their nodes to minimize certain cost functions. In the first problem, we partitioned a bipartite graph by minimizing the so-called *MinMaxCut* cost function, while in the second problem, we partitioned a structured graph by minimizing the so-called *Modified-MinMaxCut* cost function. The structural property of the graph is incorporated in defining this new cost function. The solutions we proposed are under the framework of spectral clustering, where one relies on the eigenvectors of the graph matrices, e.g., the Laplacian matrix or the adjacency matrix, and any clustering algorithm, e.g., K-means, to partition nodes into disjoint clusters.

Furthermore, for the time-variant graph, we decomposed the problem into two steps. First, we transformed the variations in the graph topology into perturbations to the graph matrices. Then we transformed the update of the clusters into an update of the (generalized) eigenvectors of these graph matrices. We utilized matrix perturbation theory to update the generalized eigenvectors and then update the clusters. Our simulations showed that on synthetic data, the proposed method can efficiently track the eigenvectors and the clusters generated by the updated eigenvectors have almost the same cost function value as that of exact computation.

Acknowledgments

功不唐捐。
All efforts are not made for nothing.

Studying abroad is never an easy thing for me, especially during the pandemic time. I would like to first pay tribute to all my efforts in the past two years. No matter what I could achieve, either good or bad, I will cherish this experience for my entire life.

I would like to express my gratitude to both my beloved and respectful supervisors. Without their help, this thesis work could not have been done. I would like to thank Mario Coutino, who is always helpful and is capable of providing me with precise and insightful comments. Working with him during the whole past year is definitely memorable. I would like to thank my supervisor Professor Geert Leus for his supervision and for giving me this thesis work opportunity as well. He has always been commenting on my work in detail and questions from him are always inspiring. Both Mario and Geert have been very patient and encouraging to me, accepting all my mistakes. It is a pity that I do not have opportunities to work with them afterward. I also want to thank Professor Huijuan Wang for her time being my thesis committee.

I would also like to express my gratitude to Tian Gan. She has been supporting me physically and mentally for the past two years. I can always talk to her when I am depressed and after each talk, I can always recover from any negative emotions. The past two years is a great journey with her accompany. I would like to thank my friend, Maosheng, for our great days in Delft, Amsterdam, Den Haag, and Rotterdam.

People in the CAS group are very kind. I would like to thank Professor Alle-Jan van der Veen for his valuable discussion on subspace tracking. I also want to thank Alberto Natali for his comments on my presentation. And I would like to mention that having Bichi and Xiaoyao as friends made my pandemic boring life colorful.

Studying at TU Delft was not possible without the scholarship from the Microelectronics department. I am grateful for the generous financial support.

Finally, I would like to express my gratitude to my parents. They always support all my decisions without any reservation and hesitation. Without them, I cannot go this far from that small city.

Yanbin He B.Sc.
Delft, The Netherlands
October 22 2021

Nomenclature

Graph

$\mathbf{W}$	Graph adjacency matrix
$\mathbf{L}$	Graph Laplacian
$\mathcal{G}(\mathcal{V}, \mathcal{E})$	Graph $\mathcal{G}$ with node set $\mathcal{V}$ and edge set $\mathcal{E}$
$\mathcal{G}_{\text{b}}(\mathcal{B},\mathcal{U}, \mathbf{B})$	Bipartite graph with node sets $\mathcal{B}, \mathcal{U}$ and connection status matrix $\mathbf{B}$
$\mathcal{G}(\mathcal{B},\mathcal{U}, \mathcal{E})$	Graph with node sets $\mathcal{B}$ and $\mathcal{U}$ and edge set $\mathcal{E}$

Linear algebra

$(\cdot)^\top$	Transpose
$(\cdot)^{-1}$	Inverse
$(\cdot)^\dagger$	Pseudoinverse
$\mathbb{R}^n$	Real vector space of n-dimensional vectors
$\mathbb{R}^{n\times n}$	Real matrices space of n by n matrices
$\mathbf{1}$	All-ones vector
$\mathbf{I}$	Identity matrix
$\text{diag}(\mathbf{a})$	Diagonal matrix with $\mathbf{a}$ on the diagonal

Mathematical Objects

a	Scalar
$\mathbf{a}$	Vector
$\mathbf{A}$	Matrix
$[\mathbf{a}]_i$	The i-th entry of vector $\mathbf{a}$
$[\mathbf{A}]_{i,j}$	Entry (i, j) of matrix $\mathbf{A}$
$[\mathbf{A}]_{i,:}$	The i-th row of matrix $\mathbf{A}$
$[\mathbf{A}]_{:,j}$	The j-th column of matrix $\mathbf{A}$

Sets

$\cup$	Union
$\cap$	Intersection
$\in$	Belong to
$\lvert\mathcal{V}\rvert$	The cardinality of set $\mathcal{V}$
$\varnothing$	Empty set

Contents

1 Introduction

Graphs have been gaining more and more attention since they can be used to model the non-Euclidean data or entities and their complicated relationships and interdependency [1]. This type of data or entities are ubiquitous and inevitable, coming from various real network applications, e.g., social media network [2], webpage and internet [3], sensor network [4], cellular network [5], recommendation system [6, 7] and so on. Furthermore, some network instances have intrinsic structures. For example, a cellular network with connections between the base stations and users can be represented as a bipartite graph model. After obtaining the data or entities and their irregular structures, some preprocessing is required such that the following processing methods can be applied. Clustering or partitioning is one of the most important preprocessing methods. By performing clustering, one can find the potential similarities within the data [8, 9], or group the entities, which can be beneficial. For example, in long-term evolution (LTE) cellular network, the capacity can be improved by doing clustering [5]. Another example can be found in a computer with multiple processors [10], where an appropriate clustering of the processors can achieve a good load balancing and minimize the ratio of communication over computation for a given task.

In the literature, various clustering techniques have been proposed so far. Among them, spectral clustering based on spectral graph theory has become one of the most popular modern clustering algorithms [11]. Spectral clustering utilizes the information carried by the eigenvectors of the Laplacian or adjacency matrix of the graph to partition the nodes of the graph [12]. Spectral clustering has a variety of advantages. First, it is simple to implement and can be solved efficiently by linear algebra methods. Second, spectral clustering can often achieve better results compared with traditional methods. However, there still exist some limitations regarding the implementation of spectral clustering. First, it suffers from a high computational complexity due to the required eigenvalue decomposition (EVD). This becomes a severe issue when the size of the graph becomes large. Second, almost all the spectral clustering algorithms are offline, which means they cannot be applied to graphs changing over time [13]. Therefore, it is necessary to design efficient methods to update the graph clusters and avoid computing the EVD from scratch every time.

Determining the boundaries and generating the clusters are making decisions. To derive appropriate and meaningful clusters, some criteria should be implemented. This can be achieved by minimizing or maximizing cost functions. Some well-known cost functions, such as *Ratio Cut* [14] and *Normalized Cut* [15], are NP-hard. It is essential to translate the cost functions into practical and tractable forms to determine the clusters. Fortunately, relaxations towards these cost functions have been widely discussed and these relaxations can be combined with the spectral clustering method [11]. The aforementioned cost functions are general

and captures metrics of interest for partition typical graphs. However, in many instances, despite the fact that the graph structure can be used to process the data defined on the nodes, it can also be used to cluster (or partition) a network in subgroups that exhibit a certain type of *closeness*, e.g., when different types of edges or nodes and consequently different connections in between are taken into account. But to relax such cost functions and produce clusters accordingly is still a challenging problem.

1.1 Research Statement

In order to face these challenges, we discuss the following topics in the thesis.

- How can we relax a specific cost function and give feasible solutions to generate the partition?
- How can we partition a structured graph changing over time adaptively and efficiently?

1.2 Outline

This thesis work is organized as follows.

- Chapter 2-Background: We give the background for the clustering problem. We first introduce the *traditional* clustering methods briefly and then go into one of the most important clustering methods, K-means. After that, we define the graph clustering problem and elaborate on the framework of spectral clustering, starting with the recursive spectral bisection. Additionally, a graph signal processing-based clustering acceleration method is introduced, which can avoid computing the eigenvalue decomposition of the Laplacian matrix.
- Chapter 3-Problem Formulation: We define the two problems that are going to be discussed and motivate these problems. The structures of these two problems are similar, and the difference is the targeted cost function. There are two stages in both problems. In the first stage, the first problem focuses on partitioning a bipartite graph and minimizing the *MinMaxCut* cost function, while in the second problem, we focus on partitioning a structured graph and minimizing the *Modified-MinMaxCut* cost function. In the second stage, we would like to design methods efficiently updating the partitions when the graphs are changing over time. We relate these two problems to some real-world cases, motivating the investigation of these problems.
- Chapter 4-Analysis of Cost Functions: We give the derivations of the relaxed cost functions and summarize them in a way similar to spectral clustering method.
- Chapter 5-Case Study: Simple Static Graph for Methods Validation and Comparison for Problem 2: In this section, we provide a simple case study regarding the devised approaches to problem 2.

- Chapter 6-Adaptive Solutions: We give the adaptive methods for partitioning the graph changing over time. Since updating the clusters is related to the update of eigenvectors, we first review some related works. After that, we give detailed derivations of efficiently updating the clusters defined by *MinMaxCut* and *Modified-MinMaxCut*.

- Chapter 7-Numerical Results: We implement the previously proposed methods. This section contains two parts. The first part is about numerical simulation of the first problem. In this part, we first introduce the way to generate the synthetic data. Then we test the methods over the synthetic data. Since in the first problem we mainly focus on updating the clusters, the simulation results are showing the adaptive clustering process. The second part includes the simulations of the second problem. We test the proposed methods on a time-varying graph, illustrating the adaptive clustering process.

- Chapter 8-Conclusion and Future Work: We conclude the thesis and propose the potential future work directions.

Bibliography

[1] Z. Wu, S. Pan, F. Chen, G. Long, C. Zhang, and S. Y. Philip, "A comprehensive survey on graph neural networks," *IEEE transactions on neural networks and learning systems*, 2020.

[2] M. A. Russell, *Mining the social web: Analyzing data from Facebook, Twitter, LinkedIn, and other social media sites.* " O'Reilly Media, Inc.", 2011.

[3] P. Desikan, N. Pathak, J. Srivastava, and V. Kumar, "Incremental page rank computation on evolving graphs," in *Special interest tracks and posters of the 14th International Conference on World Wide Web*, 2005, pp. 1094–1095.

[4] I. Jabłoński, "Graph signal processing in applications to sensor networks, smart grids, and smart cities," *IEEE Sensors Journal*, vol. 17, no. 23, pp. 7659–7666, 2017.

[5] M. Hajjar, G. Aldabbagh, and N. Dimitriou, "Using clustering techniques to improve capacity of lte networks," in *2015 21st Asia-Pacific Conference on Communications (APCC)*. IEEE, 2015, pp. 68–73.

[6] Z. Huang, W. Chung, T.-H. Ong, and H. Chen, "A graph-based recommender system for digital library," in *Proceedings of the 2nd ACM/IEEE-CS joint conference on Digital libraries*, 2002, pp. 65–73.

[7] R. Ying, R. He, K. Chen, P. Eksombatchai, W. L. Hamilton, and J. Leskovec, "Graph convolutional neural networks for web-scale recommender systems," in *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, 2018, pp. 974–983.

[8] M. Z. Rodriguez, C. H. Comin, D. Casanova, O. M. Bruno, D. R. Amancio, L. d. F. Costa, and F. A. Rodrigues, "Clustering algorithms: A comparative approach," *PloS one*, vol. 14, no. 1, p. e0210236, 2019.

[9] D. Xu and Y. Tian, "A comprehensive survey of clustering algorithms," *Annals of Data Science*, vol. 2, no. 2, pp. 165–193, 2015.

[10] Y. Saad, *Iterative methods for sparse linear systems*. SIAM, 2003.

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

[12] A. Ng, M. Jordan, and Y. Weiss, "On spectral clustering: Analysis and an algorithm," *Advances in neural information processing systems*, vol. 14, pp. 849–856, 2001.

[13] H. Ning, W. Xu, Y. Chi, Y. Gong, and T. S. Huang, "Incremental spectral clustering by efficiently updating the eigen-system," *Pattern Recognition*, vol. 43, no. 1, pp. 113–127, 2010.

[14] L. Hagen and A. B. Kahng, "New spectral methods for ratio cut partitioning and clustering," *IEEE transactions on computer-aided design of integrated circuits and systems*, vol. 11, no. 9, pp. 1074–1085, 1992.

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

2 Background

2.1 Graph

Before going into the topic, we would like to introduce a graph and define the notations.

A graph, usually denoted as $\mathcal{G}(\mathcal{V}, \mathcal{E})$, comprises two sets, $\mathcal{V}$ and $\mathcal{E}$. $\mathcal{V}$ contains all the vertices or nodes, while $\mathcal{E}$ contains all the edges or links that connect the nodes. In this thesis work, the considered graph is **undirected** and **weighted**. The term **undirected** means the edges are bidirectional, e.g., shaking hands at a party instead of transferring money. A **weighted** graph is a graph in which a number is assigned to each edge, indicating the property of the edge, such as the strength of the connection. Furthermore, we do not consider self-loops (edges that directly connect vertices to themselves) in our graphs.

A graph can be represented by its adjacency matrix $\mathbf{W} \in \mathbb{R}^{|\mathcal{V}|\times|\mathcal{V}|}$ where $[\mathbf{W}]_{i,j} = w_{i,j}$ represents the edge between nodes i and $j \in \mathcal{V}$. In this thesis work, we only consider non-negative weight. For a weighted graph, this value can be any number other than just 0 or 1. For an undirected graph, the adjacency matrix is symmetric and $[\mathbf{W}]_{i,i}$ is 0. For every node $i \in \mathcal{V}$, the *degree* $d(i)$ of node i is the sum of the weights of the edges adjacent to i

$$d(i) = \sum_{j=1}^{|\mathcal{V}|} w_{i,j}. \tag{2.1.1}$$

The *degree matrix* $\mathbf{D}$ is the diagonal matrix

$$\mathbf{D} = \mathrm{diag}(d(1), d(2), ..., d(|\mathcal{V}|)), \tag{2.1.2}$$

where $\mathrm{diag}(\cdot)$ is a function that constructs a diagonal matrix whose diagonal elements are given by its argument.

Based on the adjacency matrix $\mathbf{W}$ and the degree matrix $\mathbf{D}$, the Laplacian matrix $\mathbf{L}$ can be defined as

$$\mathbf{L} = \mathbf{D} - \mathbf{W}. \tag{2.1.3}$$

A special case of the graph is the bipartite graph. A bipartite graph is a graph whose vertices can be divided into two disjoint and independent sets, i.e., two sets of nodes without edges between them, and such that every edge connects a vertex in one of the sets to one in the other set. Let us consider a weighted undirected bipartite graph $\mathcal{G}_\mathrm{b}(\mathcal{B}, \mathcal{U}, \mathcal{E})$, where $\mathcal{B}$ and $\mathcal{U}$ are two different types of vertices. $\mathcal{E}$ is the set of all the edges within the bipartite graph. Besides the adjacency matrix $\mathbf{W}$ and the Laplacian matrix $\mathbf{L}$, a bipartite graph is closely related to another matrix $\mathbf{B}$ as well. Each entry in matrix $\mathbf{B} \in \mathbb{R}^{|\mathcal{B}|\times|\mathcal{U}|}$, e.g., $[\mathbf{B}]_{i,j} = b_{i,j}$, is the weight of the edge between vertex i in $\mathcal{B}$ and j in $\mathcal{U}$. Therefore, $\mathbf{B}$ can represent a bipartite

graph uniquely as $\mathbf{W}$ and $\mathbf{L}$ do, upto nodes relabelling. Then the notation for the bipartite graph is written as $\mathcal{G}_{\mathrm{b}}(\mathcal{B},\mathcal{U},\mathcal{E},\mathbf{B})$, or usually $\mathcal{G}_{\mathrm{b}}(\mathcal{B},\mathcal{U},\mathbf{B})$.

Another case of a graph that will be considered in this thesis work is quite similar to the bipartite graph, and it is denoted by $\mathcal{G}(\mathcal{B},\mathcal{U},\mathcal{E})$. This type of graph has two different kinds of nodes, $\mathcal{B}$ and $\mathcal{U}$. $\mathcal{E}$ contains all the edges in the graph. However, unlike the bipartite graph where an edge $e_{i,j}$ only exists between different types of vertices, in $\mathcal{G}(\mathcal{B},\mathcal{U},\mathcal{E})$, the edges also exist between the same type of nodes, thus not necessarily a bipartite graph.

2.2 Graph Clustering

Clustering, or partitioning, is a fundamental technique and has been widely discussed and implemented in various fields and areas such as data analysis including image segmentation [1–3], text mining [4] and speech separation [5, 6], unsupervised learning in machine learning [7, 8], some special systems such as document systems [9], VHDL design [10–12], and so on. The intuition behind clustering is to explore the similarity of the data as groups or to detect the unknown structure within the unlabelled data. Varieties of clustering methods have been developed, among which three categories are quite popular. They are hierarchical techniques, partitional techniques, and graph techniques [13]. Hierarchical techniques construct hierarchical relationship among data. Partitional techniques regard the center of data points as the center of the corresponding cluster [13]. Graph techniques consider each data point as a node of a graph, and the edge between them represents their relationship, e.g., Euclidean distance. In the following sections, we will review the hierarchical techniques briefly, and review the partitional techniques and graph techniques in detail since this thesis work is strongly related to the latter two techniques.

2.2.1 Hierarchical Techniques

In the hierarchical techniques, clusters are formed by a top-down or bottom-up approach [14]. The top-down approach is called divisive and the bottom-up is called agglomerative. The agglomerative approach builds clusters starting with single data points. In the beginning, each data point forms an individual cluster and then the most neighboring two clusters merge into a new cluster until all the data points are contained in one cluster or certain termination conditions are satisfied. The divisive hierarchical clustering breaks up a cluster containing all objects into smaller clusters until each object forms a cluster on its own or until it satisfies certain termination conditions. To define whether two clusters are neighboring, a few options can be tuned, such as single linkage, complete linkage, and average linkage. Different metrics will lead to different clustering results.

2.2.2 Partitional Techniques

K-means is one of the most famous clustering algorithms among the partitional techniques [15]. It is widely used to explore the potential clusters in the unlabelled data. K-means requires as the input parameters the number of clusters K and

the distance metric. The distance metric measures the distance between each pair of data points. There are several distance metric options, such as Minkowski distance, Euclidean distance, or cosine distance [13]. During the clustering process, initially, each data point is assigned to one of the randomly chosen clusters based on its distance to the center of each cluster. After assigning all the data points, K clusters have been formed and new cluster centers can be calculated again using the mean of all the data points belonging to the same cluster. And all the data points are reassociated again. These steps are repeated until no significant changes in the cluster centers can be observed.

The K-means algorithm is summarized in Algorithm 1.

Algorithm 1: K-means

Data: The number of clusters K, a set of data points $\mathbf{X}$
Result: K clusters

```
Randomly generate K initial centroids
while NotConverge do
    Calculate the distance between each node and each centroid
    Assign node to the cluster with the nearest centroid ;  /* Assignment
     step ends */
    Recalculate means (centroids) for nodes assigned to each cluster ;
     /* Update step ends */
    if Centroids don't change then
        Converge
    else
        NotConverge
    end
end
```

The main drawback of the K-means is that the results and performance of the clustering are strongly related to the choice of initial centroids [16]. This can be alleviated by running K-means several times with different centroid seeds and choosing the best one as the final result according to some criteria, e.g., inertia [17]. Or we can finely tune the initial centroids, e.g., K-means++ [18] in which one should always pick the most distant node as the next centroid.

Another example of partitional techniques is the Gaussian mixture model (GMM)-based expectation maximization (EM) algorithm [19,20]. Instead of making "hard" assignments as in K-means, GMM guesses "soft" decisions about how to assign nodes since it takes the likelihood into consideration. This algorithm assumes that all the data points are drawn from one of K Gaussian random variables depending on latent random variables [21,22]. It has two main steps. In the first step, called "E-step", the algorithm tries to guess the value of latent random variables given the data points and the current setting of parameters. After that, in the "M-step", the algorithm uses the updated value to maximize the likelihood and update the corresponding setting of parameters which are the means and the covariance matrices of the involved Gaussians. The algorithm runs until convergence and finally, the desired clusters are obtained. This algorithm has two main drawbacks. The first one is the same as for K-means. The clustering results are strongly related to the choice of initial points. The second one is that if K is not

appropriate, there will be one or some Gaussians converging to a single data point, leading to infinity in the inversion of the covariance matrix. This can be alleviated by introducing regularization, yet the technique then becomes more complicated.

2.2.3 Graph Techniques

In the graph techniques, data points are seen as nodes of a graph and the edge weights are determined by the similarity measurements between each pair of nodes. There are many options, such as the Euclidean distance, and the Gaussian similarity function [23]. After constructing the graph $\mathcal{G}(\mathcal{V}, \mathcal{E})$, partitions are defined as a set of subsets $\{\mathcal{C}_m\}_{m=1,...,K}$ where

$$\mathcal{C}_1 \cup \mathcal{C}_2 \cup ... \cup \mathcal{C}_K = \mathcal{V}, \tag{2.2.1}$$

$$\mathcal{C}_i \cap \mathcal{C}_j = \varnothing, \ \forall i, j \in \{1, ..., K\}, i \neq j. \tag{2.2.2}$$

Various types of clustering methods have been proposed, one of the most famous techniques is based on the minimal spanning tree (MST) [24]. Apart from MST, another type of graph technique called spectral clustering is widely used and discussed. In this thesis work, our solutions are strongly related to spectral clustering. Therefore, in the following sections, we will review the spectral clustering technique starting from the recursive spectral bisection solution to the multiway partition problem.

2.2.3.1 Recursive spectral bisection

Recursive spectral bisection (RSB) works on the Fiedler vector of the Laplacian matrix $\mathbf{L}$, which is the eigenvector corresponding to the second smallest eigenvalue. Let's consider a K-way clustering problem. In order to partition the graph into K clusters, RSB first partitions the graph into two subgraphs according to the entries of the Fiedler vector. This value infers the clustering information of each node and can be viewed as the coordinate of each node on a 1D number axis. RSB orders the entries of the Fiedler vector in either ascend or descend order and then assigns the nodes in the first half to one cluster and the second half to the other cluster, achieving the bisection, or it simply uses the sign of the entries and assigns the nodes which correspond to positive values to one cluster and the nodes which correspond to negative values to the other cluster. Then RSB applies bisection recursively to both the subgraphs until the number of clusters reaches K [25, 26]. The RSB is summarized in 2.

Algorithm 2: Recursive Spectral Bisection (RSB)

Data: The number of clusters K, the graph $\mathcal{G}(\mathcal{V}, \mathcal{E})$
Result: K clusters

1 Calculate the Laplacian matrix $\mathbf{L}$ of $\mathcal{G}$
2 Calculate the eigenvalue decomposition of $\mathbf{L}$, sort the Fiedler vector, e.g., in decreasing order
3 Assign the first half of nodes to $\mathcal{V}_1$, the second half to $\mathcal{V}_2$
4 Apply RSB recursively on $\mathcal{V}_1$ and $\mathcal{V}_2$ until K clusters are obtained

The RSB gives us intuition that the clustering information of nodes is embedded in the eigenvectors of the Laplacian matrix. But to obtain K clusters,

recursively applying the RSB is not straightforward. As described in [27], by including more eigenvectors and applying clustering algorithms, e.g., K-means, one can obtain K clusters in one go. This represents the framework of spectral clustering.

2.2.3.2 The Framework of Spectral Clustering

As we have discussed before, with more eigenvectors and other clustering algorithms, K clusters can be obtained directly. This is described as the spectral clustering method. [10] states that the more eigenvectors we use, the better results we can have. Usually, in a K-way partition problem, K eigenvectors are sufficient [23, 27].

Spectral clustering has the following steps. First, compute the Laplacian matrix $\mathbf{L}$ of the graph $\mathcal{G}(\mathcal{V}, \mathcal{E})$. Second, compute the eigenvalue decomposition of the Laplacian matrix $\mathbf{L}$ and pick the first K smallest eigenvectors forming $\mathbf{U}_K \in \mathbb{R}^{|\mathcal{V}| \times K}$. Third, consider each row of $\mathbf{U}_K$ as node coordinates or as spectral embeddings and apply K-means or other clustering algorithms to $\mathbf{U}_K$ and obtain the desired clusters. These steps are summarized in Algorithm 3. Usually, we have three options for designing a Laplacian matrix [28]: the general combinatorial graph Laplacian $\mathbf{L} = \mathbf{D} - \mathbf{W}$, the normalized Laplacian matrix $\mathbf{L}_n = \mathbf{D}^{-\frac{1}{2}}\mathbf{L}\mathbf{D}^{-\frac{1}{2}} = \mathbf{I} - \mathbf{D}^{-\frac{1}{2}}\mathbf{W}\mathbf{D}^{-\frac{1}{2}}$, the random walk Laplacian $\mathbf{L}_{rw} = \mathbf{D}^{-1}\mathbf{L} = \mathbf{I} - \mathbf{D}^{-1}\mathbf{W}$. A recommendation on choosing the Laplacian matrix can be found in [23].

Algorithm 3: Spectral clustering

Data: Graph $\mathcal{G}(\mathcal{V}, \mathcal{E})$ with Laplacian matrix $\mathbf{L}/\mathbf{L}_n/\mathbf{L}_{rw}$, the number of clusters K

Result: K clusters

1 Calculating the eigenvalue decomposition of Laplacian matrix
2 Choose the first K eigenvectors corresponding to K smallest eigenvalues and construct $\mathbf{U}_K$
3 Embed the i-th vertex to $[\mathbf{U}_K]_{i,:}$
4 Run K-means or any clustering algorithm to obtain the desired clusters

An illustration of the framework of spectral clustering is shown in Fig. 2.1. This is the bisection case, that is, the number of clusters K is 2. And the spectral embedding of node 1 (in red) is denoted in red as well.

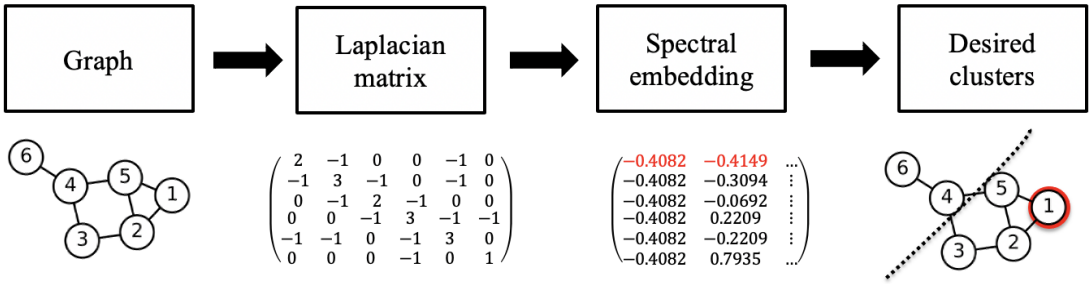


Figure 2.1: The framework of spectral clustering

2.2.3.3 Graph signal processing based algorithm

Spectral clustering is based on the eigenvectors of the Laplacian matrix, which are expensive to compute for large size graphs. Therefore, in [29], Tremblay et al. introduce an accelerated spectral clustering method by replacing the exact eigenvalue decomposition with graph filtering. The polynomial-approximated low pass filter reduces the complexity, and only matrix-vector multiplication operations are required instead of diagonalizing the Laplacian matrix.

Graph filtering is based on the graph Fourier transform. Considering graph $\mathcal{G}(\mathcal{V}, \mathcal{E})$ has Laplacian matrix $\mathbf{L}$, the eigenvalue decomposition of $\mathbf{L}$ can be written as

$$\mathbf{L} = \mathbf{U}\mathbf{\Lambda}\mathbf{U}^\top. \tag{2.2.3}$$

The $\mathbf{U}$ matrix represents the graph Fourier basis of the graph and $\mathbf{\Lambda} = \text{diag}(\lambda_1, \lambda_2, ..., \lambda_N)$ is the spectrum of the graph. If there is a signal $\mathbf{x}$ defined on this graph, then its graph Fourier transform is

$$\hat{\mathbf{x}} = \mathbf{U}^\top \mathbf{x}. \tag{2.2.4}$$

Now if we define a filter $h(\lambda)$ on $[0, \lambda_N]$, the filtered signal in the frequency domain is $\hat{\mathbf{H}}\hat{\mathbf{x}}$, where $\hat{\mathbf{H}} = \text{diag}(h(\lambda_1), h(\lambda_2), ..., h(\lambda_N))$. And the filtered signal in the graph domain is

$$\mathbf{H}\mathbf{x} = \mathbf{U}\hat{\mathbf{H}}\mathbf{U}^\top \mathbf{x}, \tag{2.2.5}$$

where the $\mathbf{H} \in \mathbb{R}^{|\mathcal{V}|\times|\mathcal{V}|}$ is the associate filter operator, defined as

$$\mathbf{H} = \mathbf{U}\hat{\mathbf{H}}\mathbf{U}^\top. \tag{2.2.6}$$

If the filter $h(\lambda)$ can be approximated by a polynomial using $h(\lambda) \simeq \sum_{l=0}^{m} \alpha_l \lambda^l$, Equation 2.2.5 can be rewritten as

$$\mathbf{H}\mathbf{x} = \mathbf{U}\hat{\mathbf{H}}\mathbf{U}^\top \mathbf{x} \simeq \sum_{l=0}^{m} \alpha_l \mathbf{L}^l \mathbf{x}. \tag{2.2.7}$$

In such a way, the eigenvalue decomposition is avoided. This fast filtering method has complexity $O(m|\mathcal{E}| + |\mathcal{V}|)$ where m is the order of the polynomial of the filter.

Specifically, if the $h(\lambda)$ is an ideal low pass filter defined on $[0, \lambda_K]$, where K is the number of clusters, the corresponding graph filter operator $\mathbf{H}_{\lambda_K}$ is

$$\mathbf{H}_{\lambda_K} = \mathbf{U} \begin{pmatrix} \mathbf{I}_K & 0 \\ 0 & 0 \end{pmatrix} \mathbf{U}^\top = \mathbf{U}_K \mathbf{U}_K^\top. \tag{2.2.8}$$

We can use this filter operator to filter η random signals $\mathbf{r}_i$, whose components are independent random Gaussian variables of zero-mean and variance $\frac{1}{\eta}$, each row of the filtered signals $\mathbf{H}_{\lambda_K}\mathbf{R} \in \mathbb{R}^{|\mathcal{V}|\times\eta}$, where $\mathbf{R} = [\mathbf{r}_1, \ldots, \mathbf{r}_\eta]$, can be seen as the spectral embedding of the nodes in the graph as well.

Proposition 1. Let ϵ, β>0 be given. If η is larger than

$$\eta_0 = \frac{4+2\beta}{\epsilon^2/2 - \epsilon^3/3} \log N, \tag{2.2.9}$$

with probability at least $1 - N^{-\beta}$, we have: $\forall (i,j) \in [1,N]^2$

$$(1-\epsilon)||\mathbf{f}_i - \mathbf{f}_j||^2 \leq ||\tilde{\mathbf{f}}_i - \tilde{\mathbf{f}}_j||^2 \leq (1+\epsilon)||\mathbf{f}_i - \mathbf{f}_j||^2, \tag{2.2.10}$$

where $\mathbf{f}_i$ is the spectral embedding (row of $\mathbf{U}_K$) via eigenvalue decomposition, $\tilde{\mathbf{f}}_i$ is the approximated one, via the filtered Gaussians, and the row of $\mathbf{H}_{\lambda_K}\mathbf{R}$.

Proof. See [29] □

This proposition means the distance between coordinates of different nodes introduced in Background is preserved and approximated in the filtered signals, that is, each row of the filtered signal can be seen as the coordinates or the spectral embedding of vertices as well. To conclude, the graph signal processing based spectral clustering method has a total of 4 steps [29,30]. The first step is to estimate the largest and K-th smallest eigenvalue of the Laplacian matrix. The K-th smallest eigenvalue estimation is based on the eigencount techniques. In the second step, one should first define an ideal low pass filter with cut-off frequency λ_K. The ideal low pass filter is approximated by a Chebychev polynomial with specified order m. Then the filtering operator is obtained. As we can imagine, higher order polynomials can bring higher accuracy but require more computation. The third step is generating η Gaussian random signals with variance $\frac{1}{\eta}$. Then $\mathbf{R} \in \mathbb{R}^{|\mathcal{V}|\times\eta}$ is filtered by a graph filtering operator. In the fourth step, one can apply K-means or any other clustering algorithm on the filtered signals and to obtain the desired clusters.

Bibliography

[1] M. Belkin and P. Niyogi, "Laplacian eigenmaps for dimensionality reduction and data representation," *Neural computation*, vol. 15, no. 6, pp. 1373–1396, 2003.

[2] M. Meila and J. Shi, "Learning segmentation by random walks," *Advances in neural information processing systems*, vol. 13, pp. 873–879, 2000.

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

[4] C. Ding, X. He, and H. D. Simon, "On the equivalence of nonnegative matrix factorization and spectral clustering," in *Proceedings of the 2005 SIAM international conference on data mining*. SIAM, 2005, pp. 606–610.

[5] F. R. Bach and M. I. Jordan, "Learning spectral clustering, with application to speech separation," *The Journal of Machine Learning Research*, vol. 7, pp. 1963–2001, 2006.

[6] S. Furui, "Unsupervised speaker adaptation based on hierarchical spectral clustering," *IEEE Transactions on Acoustics, Speech, and Signal Processing*, vol. 37, no. 12, pp. 1923–1930, 1989.

[7] Z. Ghahramani, "Unsupervised learning," in *Summer School on Machine Learning*. Springer, 2003, pp. 72–112.

[8] M. Caron, P. Bojanowski, A. Joulin, and M. Douze, "Deep clustering for unsupervised learning of visual features," in *Proceedings of the European Conference on Computer Vision (ECCV)*, 2018, pp. 132–149.

[9] Y. Zhao, G. Karypis, and U. Fayyad, "Hierarchical clustering algorithms for document datasets," *Data mining and knowledge discovery*, vol. 10, no. 2, pp. 141–168, 2005.

[10] C. J. Alpert and S.-Z. Yao, "Spectral partitioning: The more eigenvectors, the better," in *Proceedings of the 32nd annual ACM/IEEE design automation conference*, 1995, pp. 195–200.

[11] M. López-Vallejo and J. C. López, "On the hardware-software partitioning problem: System modeling and partitioning techniques," *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, vol. 8, no. 3, pp. 269–297, 2003.

[12] P. Arató, Z. A. Mann, and A. Orbán, "Algorithmic aspects of hardware/software partitioning," *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, vol. 10, no. 1, pp. 136–156, 2005.

[13] D. Xu and Y. Tian, "A comprehensive survey of clustering algorithms," *Annals of Data Science*, vol. 2, no. 2, pp. 165–193, 2015.

[14] F. Murtagh and P. Contreras, "Algorithms for hierarchical clustering: an overview," *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, vol. 2, no. 1, pp. 86–97, 2012.

[15] M. Z. Rodriguez, C. H. Comin, D. Casanova, O. M. Bruno, D. R. Amancio, L. d. F. Costa, and F. A. Rodrigues, "Clustering algorithms: A comparative approach," *PloS one*, vol. 14, no. 1, p. e0210236, 2019.

[16] J. M. Pena, J. A. Lozano, and P. Larranaga, "An empirical comparison of four initialization methods for the k-means algorithm," *Pattern recognition letters*, vol. 20, no. 10, pp. 1027–1040, 1999.

[17] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[18] D. Arthur and S. Vassilvitskii, "k-means++: The advantages of careful seeding," Stanford, Tech. Rep., 2006.

[19] C. Fraley and A. E. Raftery, "How many clusters? which clustering method? answers via model-based cluster analysis," *The computer journal*, vol. 41, no. 8, pp. 578–588, 1998.

[20] A. Saxena, M. Prasad, A. Gupta, N. Bharill, O. P. Patel, A. Tiwari, M. J. Er, W. Ding, and C.-T. Lin, "A review of clustering techniques and developments," *Neurocomputing*, vol. 267, pp. 664–681, 2017.

[21] A. P. Dempster, N. M. Laird, and D. B. Rubin, "Maximum likelihood from incomplete data via the em algorithm," *Journal of the Royal Statistical Society: Series B (Methodological)*, vol. 39, no. 1, pp. 1–22, 1977.

[22] A. Ng, "Cs229 lecture notes," *CS229 Lecture notes*, vol. 1, no. 1, pp. 1–3, 2000.

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

[24] C. T. Zahn, "Graph-theoretical methods for detecting and describing gestalt clusters," *IEEE Transactions on computers*, vol. 100, no. 1, pp. 68–86, 1971.

[25] Y. Saad, *Iterative methods for sparse linear systems*. SIAM, 2003.

[26] H. Zha, X. He, C. Ding, H. Simon, and M. Gu, "Bipartite graph partitioning and data clustering," in *Proceedings of the tenth international conference on Information and knowledge management*, 2001, pp. 25–32.

[27] A. Ng, M. Jordan, and Y. Weiss, "On spectral clustering: Analysis and an algorithm," *Advances in neural information processing systems*, vol. 14, pp. 849–856, 2001.

[28] F. R. Chung and F. C. Graham, *Spectral graph theory*. American Mathematical Soc., 1997, no. 92.

[29] N. Tremblay, G. Puy, P. Borgnat, R. Gribonval, and P. Vandergheynst, "Accelerated spectral clustering using graph filtering of random signals," in *2016 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. Ieee, 2016, pp. 4094–4098.

[30] J. Paratte and L. Martin, "Fast eigenspace approximation using random signals," *arXiv preprint arXiv:1611.00938*, 2016.

3 Problem Formulation

In this section, we are going to elaborate on the details of the research topics in this thesis work. First, we define some functions as preliminary knowledge.

Let us consider a graph $\mathcal{G}(\mathcal{V}, \mathcal{E})$, with the adjacency matrix $\mathbf{W}$ and a set of subsets of nodes $\{\mathcal{C}_m\}_{m=1,...,K}$ where $\mathcal{C}_1 \cup \mathcal{C}_2 \cup ... \cup \mathcal{C}_K = \mathcal{V}$ and $\mathcal{C}_i \cap \mathcal{C}_j = \varnothing$, $\forall i, j \in \{1, ..., K\}, i \neq j$. An interconnection function between two subsets $\mathcal{C}_p$ and $\mathcal{C}_q$ is defined as

$$W(\mathcal{C}_p, \mathcal{C}_q) = \sum_{i \in \mathcal{C}_p, j \in \mathcal{C}_q} w_{i,j}, \tag{3.0.1}$$

where $w_{i,j} = [\mathbf{W}]_{i,j}$.

The *cut* of a subgraph is defined as

$$cut(\mathcal{C}_m) = \sum_{i \in \mathcal{C}_m, j \in \mathcal{V} \setminus \mathcal{C}_m} w_{i,j}, \tag{3.0.2}$$

where $\mathcal{V} \setminus \mathcal{C}_m$ is the complement of $\mathcal{C}_m$.

The *vol* of a subgraph is defined as

$$vol(\mathcal{C}_m) = \sum_{i \in \mathcal{C}_m, j \in \mathcal{V}} w_{i,j}. \tag{3.0.3}$$

Based on these quantities, we define later metrics for partition graphs in this thesis.

3.1 Problem 1

The first problem is the so-called bipartite graph clustering problem and the targeted graph has bipartite structure. A bipartite graph is a graph whose vertices can be divided into two disjoint and independent sets and such that every edge connects a vertex in one of the sets to one in the other set. Let us consider a weighted undirected bipartite graph $\mathcal{G}_{\mathrm{b}}(\mathcal{B}, \mathcal{U}, \mathbf{B})$, where $\mathcal{B}$ and $\mathcal{U}$ are two different types of vertices. $\mathcal{E}$ is the set of all the edges within the bipartite graph. $\mathbf{B}$ contains the weights of all the edges, as we have defined before.

A bipartite graph can be seen as a model for many systems, such as a documents-terms system [1–3], a communication system [4], etc. In our case, we consider the bipartite graph is the model for a cellular network. In this setting, $\mathcal{B}$ is the set of base stations and $\mathcal{U}$ is the set of users. The reason for investigating the bipartite graph clustering problem in a cellular network is that appropriate partitions can achieve a good load balancing [5] or they can minimize the interference and communication cost among different regional clusters [6]. In the rest of this report, the word "base station" and "user" will refer to the vertices in $\mathcal{B}$

and $\mathcal{U}$, respectively. We assume here that the number of users $|\mathcal{U}|$ is much larger than the number of base stations $|\mathcal{B}|$.

As we have discussed before, the partitions should satisfy certain criteria. In the first problem, the desired partitions are defined by the following problem statement:

- Given the number of clusters K, decompose the vertices within the bipartite graph into K disjoint and non-empty clusters $\{\mathcal{C}_m\}_{m=1,\dots,K}$ by solving the following optimization problem

$$\min_{\{\mathcal{C}_m\}} J_1(\{\mathcal{C}_m\}) \tag{3.1.1}$$

$$=\min_{\{\mathcal{C}_m\}} \sum_{m=1}^{K} \frac{cut(\mathcal{C}_m)}{vol(\mathcal{C}_m) - cut(\mathcal{C}_m)} \tag{3.1.2}$$

$$=\min_{\{\mathcal{C}_m\}} \sum_{m=1}^{K} \frac{cut(\mathcal{C}_m)}{W(\mathcal{C}_m, \mathcal{C}_m)}. \tag{3.1.3}$$

This cost function is called *MinMaxCut*. If we look into the cost function, the numerator of each term is the sum of all the cut edges, which can be seen as the interference of the connection or the traffic routed between different subgraphs, because it models the connection starting from the node $i \in \mathcal{C}_m$ to the node $j \notin \mathcal{C}_m$. The denominator of each term contains the sum of the edge weights within each subgraph, which can be viewed as the within-cluster similarities or the density of the subgraph, e.g., traffic routed within the clusters. Consequently, minimizing this cost function can achieve the following aspects. First, it can generate clusters with a small number of cut edges, minimizing the communications within different clusters. Second, it enforces the subgraphs to be as dense and similar as possible, trying to contain as much traffic as possible within the cluster. Additionally, revealed by [3, 5], it can avoid generating unbalanced partitions compared with other partition criteria such as *Ratio Cut*, which is preferable as well.

In a practical scenario, the cellular network is dynamic due to the mobility of users. Apart from that, new users may come into the network or users may leave the network, changing the bipartite graph as well. Furthermore, the update of the network structure, i.e., the installation of new base stations will also change the graph. When an update of the clusters is required, one can certainly opt to recompute the eigenvalue decomposition from scratch. However, for a large cellular network, this is expensive to do so. Therefore, it is necessary to design a method that is capable of updating the clusters when the graph is changing over time. This is one of the main aspects we will investigate.

3.2 Problem 2

In the second problem, we focus on a quite similar problem to the first one, but with slightly different graph settings and cost function.

Let us consider a graph $\mathcal{G}(\mathcal{B},\mathcal{U},\mathcal{E})$. Remember that there are *two* types of vertices $\mathcal{B}$ and $\mathcal{U}$ in the graph, satisfying

$$\mathcal{B} \cup \mathcal{U} = \mathcal{V} \text{ and } \mathcal{B} \cap \mathcal{U} = \varnothing,$$

where $|\mathcal{V}| = N$.

Unlike the bipartite graph, an edge $e_{i,j} \in \mathcal{E}$ in this case can exist between the *same* and *different* types of vertices $\mathcal{B}$ and $\mathcal{U}$ with edge weight $w_{i,j} \geq 0$. Thus, this graph is *not* bipartite.

In this problem, the desired partitions are defined by the following problem statement:

- Given the number of clusters K, the objective is to decompose the vertices into K disjoint and non-empty clusters $\{\mathcal{C}_m\}_{m=1,\dots,K}$ by solving the following optimization problem

$$\min_{\{\mathcal{C}_m\}} \; J_2(\{\mathcal{C}_m\}) \tag{3.2.1}$$

$$= \min_{\{\mathcal{C}_m\}} \sum_{m=1}^{K} \frac{\sum_{i\in\mathcal{C}_m, j\in\mathcal{V}\setminus\mathcal{C}_m} w_{i,j}}{\sum_{i\in\mathcal{B}_m, j\in\mathcal{U}_m} w_{i,j}} \tag{3.2.2}$$

$$= \min_{\{\mathcal{C}_m\}} \sum_{m=1}^{K} \frac{cut(\mathcal{C}_m)}{W(\mathcal{B}_m,\mathcal{U}_m)}, \tag{3.2.3}$$

  where $\mathcal{B}_m = \mathcal{C}_m \cap \mathcal{B}(\neq \varnothing)$ and $\mathcal{U}_m = \mathcal{C}_m \cap \mathcal{U}(\neq \varnothing)$.

As we can see, the cost function (3.2.3) is almost the same as (3.1.3) except for the denominator. For the sake of convenience, we call it *Modified-MinMaxCut*. The denominator of each term in the cost function (3.1.3) implies that the density or the within-cluster similarity is only defined on the edges between different types of nodes. As we can see, the main difference between this cost function and the one in the first problem is that this cost function incorporates the graph structure, defining clusters only based on one type of edge. Function (3.2.3) enables us to work on a graph that has not only two different types of nodes but also two different types of edges, broadening the model of a bipartite graph. One type of the edges can be the wireless links in the cellular network and the other type can be the wired links. For example, in a 4G/5G cellular network, when base station cooperation or the Coordinated Multipoint (CoMP) concept is implemented to provide the cell edge users with better performance, some information has to be exchanged between different base stations [7–11]. This type of information, including channel state information (CSI) or user data, is exchanged through the backhaul network. However, in the real world, exchanging user data or CSI within the whole network is infeasible since users are many and data are increasing rapidly, leading to a signaling overhead. Therefore, in order to resolve this problem, the information can be exchanged locally [12,13], that is, identifying the clusters and the exchange is performed within each cluster. We want to minimize the operation cost, as we have motivated that traffic routed between different clusters produces high cost, and both traffic in the wired and wireless link should be taken into consideration,

while in the denominator, from the perspective of interference between different clusters, there is no need to consider the wired link since the interference caused by wired link is ignorable. That is, we want the clusters to be as dense as possible in the sense of wireless links, but in the meantime, we also want to reduce the cost of routing traffic over both wired and wireless links between different clusters. As we can see, in this case, we have to partition a graph in which we have two types of edges which are wireless links and links from a backhaul network, motivating the problem. When one cares about the number of cut edges between the different clusters but doesn't want to consider the wired links in the density, the cost (3.2.3) is preferred.

Furthermore, a time-varying solution towards (3.2.3) is essential as well. Therefore, our research not only focuses on (3.2.3) by giving a tractable solution, but also on providing a method that can update the clusters adaptively and efficiently.

To conclude, the main drivers of this thesis are:

- Give a tractable solution for the graph partition problems involving the cost function (3.1.3) over *a static bipartite graph* and for the cost function (3.2.3) over *a structured static graph*
- Derive methods that can adaptively update the partitions defined by (3.1.3) and (3.2.3), namely, partition time-varying graphs

Bibliography

[1] C. H. Ding, X. He, H. Zha, M. Gu, and H. D. Simon, "A min-max cut algorithm for graph partitioning and data clustering," in *Proceedings 2001 IEEE international conference on data mining*. IEEE, 2001, pp. 107–114.

[2] I. S. Dhillon, "Co-clustering documents and words using bipartite spectral graph partitioning," in *Proceedings of the seventh ACM SIGKDD international conference on Knowledge discovery and data mining*, 2001, pp. 269–274.

[3] H. Zha, X. He, C. Ding, H. Simon, and M. Gu, "Bipartite graph partitioning and data clustering," in *Proceedings of the tenth international conference on Information and knowledge management*, 2001, pp. 25–32.

[4] A. Engels, M. Reyer, A. Steiger, and R. Mathar, "Min-cut based partitioning for urban lte cell site planning," in *2013 IEEE 10th Consumer Communications and Networking Conference (CCNC)*. IEEE, 2013, pp. 515–520.

[5] Y. Saad, *Iterative methods for sparse linear systems*. SIAM, 2003.

[6] A. Lisser and F. Rendl, "Graph partitioning using linear and semidefinite programming," *Mathematical Programming*, vol. 95, no. 1, pp. 91–101, 2003.

[7] J. Zhao, T. Q. Quek, and Z. Lei, "Coordinated multipoint transmission with limited backhaul data transfer," *IEEE Transactions on Wireless Communications*, vol. 12, no. 6, pp. 2762–2775, 2013.

[8] T. Biermann, L. Scalia, C. Choi, H. Karl, and W. Kellerer, "Comp clustering and backhaul limitations in cooperative cellular mobile access networks," *Pervasive and Mobile Computing*, vol. 8, no. 5, pp. 662–681, 2012.

[9] J.-M. Moon and D.-H. Cho, "Inter-cluster interference management based on cell-clustering in network mimo systems," in *2011 IEEE 73rd Vehicular Technology Conference (VTC Spring)*. IEEE, 2011, pp. 1–6.

[10] C. T. Ng and H. Huang, "Linear precoding in cooperative mimo cellular networks with limited coordination clusters," *IEEE Journal on Selected Areas in communications*, vol. 28, no. 9, pp. 1446–1454, 2010.

[11] S. Bassoy, M. Jaber, M. A. Imran, and P. Xiao, "Load aware self-organising user-centric dynamic comp clustering for 5g networks," *IEEE Access*, vol. 4, pp. 2895–2906, 2016.

[12] J. Zhao and Z. Lei, "Clustering methods for base station cooperation," in *2012 IEEE Wireless Communications and Networking Conference (WCNC)*. IEEE, 2012, pp. 946–951.

[13] S. Bassoy, H. Farooq, M. A. Imran, and A. Imran, "Coordinated multi-point clustering schemes: A survey," *IEEE Communications Surveys & Tutorials*, vol. 19, no. 2, pp. 743–764, 2017.

4 Analysis of Cost Functions

In this chapter, we provide relaxations of the cost functions introduced in the previous chapter and devise tractable solutions for the static graph multiway clustering problem.

4.1 Bipartite Graph Partitioning defined by MinMaxCut

4.1.1 Relaxation of the Cost Function

For the sake of recollection, here the cost function for the graph $\mathcal{G}_{\mathrm{b}}(\mathcal{B},\mathcal{U},\mathbf{B})$ is provided again

$$\min_{\{\mathcal{C}_m\}} J_1(\{\mathcal{C}_m\}) \tag{4.1.1}$$

$$=\min_{\{\mathcal{C}_m\}} \sum_{m=1}^{K} \frac{cut(\mathcal{C}_m)}{vol(\mathcal{C}_m) - cut(\mathcal{C}_m)} \tag{4.1.2}$$

$$=\min_{\{\mathcal{C}_m\}} \sum_{m=1}^{K} \frac{cut(\mathcal{C}_m)}{W(\mathcal{C}_m,\mathcal{C}_m)}, \tag{4.1.3}$$

where $\mathcal{C}_1 \cup \mathcal{C}_2 \cup ... \cup \mathcal{C}_K = \mathcal{V}$ and $\mathcal{C}_i \cap \mathcal{C}_j = \varnothing,\ \forall i,j \in \{1,...,K\}, i \neq j$.

It is well-known that this cost function can be reformulated as the following summation of Rayleigh quotients [1–6]

$$J_1(\{\mathcal{C}_m\}) = \sum_{m=1}^{K} \frac{\mathbf{c}_m^\top \mathbf{L} \mathbf{c}_m}{\mathbf{c}_m^\top \mathbf{W} \mathbf{c}_m}, \tag{4.1.4}$$

where

- $\mathbf{c}_m \in \{0,1\}^{|\mathcal{B}\cup\mathcal{U}|}$: indicator vector for set $\mathcal{C}_m$, and $\sum_{m=1}^{K} \mathbf{c}_m = \mathbf{1}$,
- $\mathbf{W} \in \mathbb{R}^{|\mathcal{B}\cup\mathcal{U}|\times|\mathcal{B}\cup\mathcal{U}|}$: the weighted adjacency matrix,
- $\mathbf{L} \in \mathbb{R}^{|\mathcal{B}\cup\mathcal{U}|\times|\mathcal{B}\cup\mathcal{U}|}$: the Laplacian matrix.

To provide an illustrative example of the indicator vector, let us consider the graph bisection in Fig. 2.1. Node 4 and 6 are in the left cluster while the rest are in the right one. Therefore, by setting corresponding entries to binary values, we can indicate the relationship between each node and each subgraph, which forms the *indicator vectors* for subgraphs.

$$\mathbf{c}_{\mathrm{l}} = [0,0,0,1,0,1]^\top \tag{4.1.5}$$

$$\mathbf{c}_{\mathrm{r}} = [1,1,1,0,1,0]^\top \tag{4.1.6}$$

Since the variable is discrete, this optimization problem is NP-hard due to the combinatoric nature of the problem [4]. However, it can be shown that if the binary constraint on the indicator vector $\mathbf{c}_m$ is relaxed to the case where the entries of $\mathbf{c}_m$ can take continuous values, then the continuous solutions towards this problem can be determined by the following generalized eigenvalue problem (GEVP)

$$\mathbf{L}\mathbf{c}_m = \lambda_m \mathbf{D}\mathbf{c}_m, \tag{4.1.7}$$

where $\mathbf{D}$ is the degree matrix of the graph.

Proposition 2. The continuous solutions (continuous indicator vectors) related to minimizing (4.1.4) are the K eigenvectors of (4.1.7) corresponding to the K smallest eigenvalues.

Proof. See [1, 2] □

However, the true solutions should be binary indicator vectors since we still need to assign nodes to different subgraphs. Therefore, a step to make the eigenvectors discrete is required, which is summarized as follows. $\{\mathbf{c}_m\}_{m=1,...,K}$ are the K generalized eigenvectors related to the K largest eigenvalues. In order to obtain K discrete binary indicator vectors, one can first stack $\{\mathbf{c}_m\}_{m=1,...,K}$ into $\mathbf{U}_K = [\mathbf{c}_1, \mathbf{c}_2, ..., \mathbf{c}_K]$, then treat each row of $\mathbf{U}_K$ as the coordinate or the spectral embedding of each node and finally apply any clustering algorithm, such as K-means.

As we can notice, the solutions are related to the framework of spectral clustering when we choose the eigenvectors of the normalized Laplacian $\mathbf{L}_n = \mathbf{D}^{-\frac{1}{2}}\mathbf{L}\mathbf{D}^{-\frac{1}{2}}$. The continuous solutions can be retrieved by left multiplying $\mathbf{D}^{-\frac{1}{2}}$ with the eigenvectors of the normalized Laplacian $\mathbf{L}_n$.

4.1.2 Multiway Partition in Bipartite Graph

When the graph we are working on is not a general graph but a bipartite graph $\mathcal{G}_\mathrm{b}(\mathcal{B}, \mathcal{U}, \mathbf{B})$, the GEVP (4.1.7) related to the optimization problem (4.1.4) can be rewritten. In the following section, we will review the multiway partition for the bipartite graph. Here, we first recall the definition of the $\mathbf{B}$ matrix that indicates the connection status between the vertices in the set of base stations and users:

- $\mathbf{B} \in \mathbb{R}^{|\mathcal{B}|\times|\mathcal{U}|}$, $[\mathbf{B}]_{i,j} = b_{i,j}$ is the weight of the edge between vertex i in $\mathcal{B}$ and j in $\mathcal{U}$. In a cellular network, this value can be any metric defining the link strength.

Proposition 3. The adjacency matrix $\mathbf{W}$ of a bipartite graph has the following structure

$$\mathbf{W} = \mathbf{D} - \mathbf{L} \tag{4.1.8}$$

$$= \begin{bmatrix} \mathbf{0} & \mathbf{B} \\ \mathbf{B}^\top & \mathbf{0} \end{bmatrix}. \tag{4.1.9}$$

If we plug this back into the previous GEVP (4.1.7), we will arrive at the continuous multiway partitioning solution in the bipartite case, which is defined on $\mathbf{B}$. And the corresponding $\mathbf{B}$-based solution is summarized in the following proposition.

Proposition 4. Let $\mathbf{b}_m$ and $\mathbf{u}_m$ represent the continuous indicator vectors for base stations and users in subgraph $\mathcal{C}_m$, respectively, which are related to the continuous solutions of (4.1.4) for a bipartite graph..

$$\mathbf{B}\mathbf{u}_m = \mu_m \mathbf{D}_{\mathrm{b}} \mathbf{b}_m \tag{4.1.10}$$

$$\mathbf{B}^\top \mathbf{b}_m = \mu_m \mathbf{D}_{\mathrm{u}} \mathbf{u}_m \tag{4.1.11}$$

Proof. If we replace $\mathbf{L}$ with $\mathbf{D} - \mathbf{W}$ and reorganize the GEVP (4.1.7), we obtain another GEVP

$$\mathbf{W}\mathbf{c}_m = (1 - \lambda_m)\mathbf{D}\mathbf{c}_m. \tag{4.1.12}$$

Here we can fully exploit the special structure of the adjacency matrix for a bipartite graph. Let us define $\mathbf{c}_m = \begin{bmatrix} \mathbf{b}_m \\ \mathbf{u}_m \end{bmatrix}$, $\mathbf{D} = \begin{bmatrix} \mathbf{D}_{\mathrm{b}} & \mathbf{0} \\ \mathbf{0} & \mathbf{D}_{\mathrm{u}} \end{bmatrix}$, $\mathbf{D}_{\mathrm{b}} = \mathrm{diag}(\mathbf{B}\mathbf{1})$, $\mathbf{D}_{\mathrm{u}} = \mathrm{diag}(\mathbf{B}^\top \mathbf{1})$, then we obtain

$$\mathbf{B}\mathbf{u}_m = \mu \mathbf{D}_{\mathrm{b}} \mathbf{b}_m, \tag{4.1.13}$$

$$\mathbf{B}^\top \mathbf{b}_m = \mu \mathbf{D}_{\mathrm{u}} \mathbf{u}_m, \tag{4.1.14}$$

□

These vectors $\mathbf{b}_m$ and $\mathbf{u}_m$ can be computed by the following proposition.

Proposition 5. Define $\tilde{\mathbf{B}} = \mathbf{D}_{\mathrm{b}}^{-\frac{1}{2}} \mathbf{B} \mathbf{D}_{\mathrm{u}}^{-\frac{1}{2}}$, then if the left and right singular vectors of $\tilde{\mathbf{B}}$ corresponding to the m-th largest singular value are $\tilde{\mathbf{b}}_m$ and $\tilde{\mathbf{u}}_m$, we have

$$\mathbf{b}_m = \mathbf{D}_{\mathrm{b}}^{-\frac{1}{2}} \tilde{\mathbf{b}}_m, \tag{4.1.15}$$

$$\mathbf{u}_m = \mathbf{D}_{\mathrm{u}}^{-\frac{1}{2}} \tilde{\mathbf{u}}_m. \tag{4.1.16}$$

Proof. Define $\tilde{\mathbf{b}}_m = \mathbf{D}_{\mathrm{b}}^{\frac{1}{2}} \mathbf{b}_m$, $\tilde{\mathbf{u}}_m = \mathbf{D}_{\mathrm{u}}^{\frac{1}{2}} \mathbf{u}_m$, then from (4.1.13) and (4.1.14), we have

$$\mathbf{B}\mathbf{D}_{\mathrm{u}}^{-\frac{1}{2}} \tilde{\mathbf{u}}_m = \mu_m \mathbf{D}_{\mathrm{b}}^{\frac{1}{2}} \tilde{\mathbf{b}}_m, \tag{4.1.17}$$

$$\mathbf{B}^\top \mathbf{D}_{\mathrm{b}}^{-\frac{1}{2}} \tilde{\mathbf{b}}_m = \mu_m \mathbf{D}_{\mathrm{u}}^{\frac{1}{2}} \tilde{\mathbf{u}}_m. \tag{4.1.18}$$

Left multiplying with $\mathbf{D}_{\mathrm{b}}^{-\frac{1}{2}}$ and $\mathbf{D}_{\mathrm{u}}^{-\frac{1}{2}}$

$$\mathbf{D}_{\mathrm{b}}^{-\frac{1}{2}} \mathbf{B} \mathbf{D}_{\mathrm{u}}^{-\frac{1}{2}} \tilde{\mathbf{u}}_m = \mu_m \tilde{\mathbf{b}}_m, \tag{4.1.19}$$

$$\mathbf{D}_{\mathrm{u}}^{-\frac{1}{2}} \mathbf{B}^\top \mathbf{D}_{\mathrm{b}}^{-\frac{1}{2}} \tilde{\mathbf{b}}_m = \mu_m \tilde{\mathbf{u}}_m, \tag{4.1.20}$$

which shows that $\tilde{\mathbf{b}}_m$ and $\tilde{\mathbf{u}}_m$ are exactly the left and right singular vectors of $\tilde{\mathbf{B}}$ with singular value μ_m. And by left multiplying $\mathbf{D}_{\mathrm{b}}^{-\frac{1}{2}}$ and $\mathbf{D}_{\mathrm{u}}^{-\frac{1}{2}}$ to $\tilde{\mathbf{b}}_m$ and $\tilde{\mathbf{u}}_m$, we can retrieve $\mathbf{b}_m$ and $\mathbf{u}_m$. □

So far, we have given the modified GEVP in the case of a bipartite graph. All the propositions will lead to the spectral clustering method for a bipartite graph presented in Algorithm 4 [3].

Algorithm 4: Bipartite Spectral Clustering

Data: The number of clusters K, the graph $\mathcal{G}_{\mathrm{b}}(\mathcal{B},\mathcal{U},\mathbf{B})$
Result: K clusters
1 Compute $\tilde{\mathbf{B}} = \mathbf{D}_{\mathrm{b}}^{-\frac{1}{2}}\mathbf{B}\mathbf{D}_{\mathrm{u}}^{-\frac{1}{2}}$
2 Compute the left and right singular vectors of $\tilde{\mathbf{B}}$ corresponding to the K largest singular values, represented by $\tilde{\mathbf{b}}_m$ and $\tilde{\mathbf{u}}_m$, and stack them into the matrices $\tilde{\mathbf{U}}_K$ and $\tilde{\mathbf{V}}_K$
3 Treat each row of $\mathbf{U}_K = \mathbf{D}_{\mathrm{b}}^{-\frac{1}{2}}\tilde{\mathbf{U}}_K$ and $\mathbf{V}_K = \mathbf{D}_{\mathrm{u}}^{-\frac{1}{2}}\tilde{\mathbf{V}}_K$ as coordinates and jointly apply K-means

4.1.3 Fast Bipartite Graph Clustering

In this section, we will propose a fast bipartite graph partitioning method called *user assignment method*. This fast partitioning method follows the idea that we can partition the whole bipartite graph by first partitioning a subset of the original graph. In our case, we can first partition the base stations and then assign the users to the clustered base stations according to the weight of the edge between a user and a base station. Partitioning only the base stations can rely on the matrix $\mathbf{U}_K$ introduced in Algorithm 4, which can be obtained by a singular value decomposition or power method. The base station clusters are then obtained by performing K-means on this matrix. The importance of this method will be clearer after discussing the method for updating $\mathbf{U}_K$ if the bipartite graph is varying, namely, the adaptive version of the fast bipartite graph partition method. Here, we are going to elaborate on the user assignment method.

$$\begin{aligned} m^* = \arg\max_{m} \quad & \omega_{\mathcal{B}_m,j} \\ \text{s.t.} \quad & \omega_{\mathcal{B}_m,j} = \textstyle\sum_{i\in\mathcal{B}_m} b_{i,j} \end{aligned} \tag{4.1.21}$$

Assume that we have already obtained the clusters for base stations $\{\mathcal{B}_m\}_{m=1,\dots,K}$. We would like to assign user j to one of the clusters. Here we first introduce a term: aggregated edge. The aggregated edge is the edge indicating the connection between a user j and a base station cluster m and its weight is the sum of the weights of all the edges between the user j and the base stations in the cluster m. We can now simply assign the user to the cluster which has the highest weight of the aggregated edge. In other words, this method assigns user j in $\mathcal{U}$ to cluster $\mathcal{B}_{m^*}$ if the aggregated weight between u and $\mathcal{B}_m$ is maximized for cluster m^*. This is the same as solving the problem (4.1.21). The fast bipartite

graph clustering method is summarized in Algorithm 5.

Algorithm 5: Fast Bipartite Graph Clustering

Data: The number of clusters K, the graph $\mathcal{G}_{\mathrm{b}}(\mathcal{B},\mathcal{U},\mathbf{B})$
Result: K clusters
Compute $\tilde{\mathbf{B}} = \mathbf{D}_{\mathrm{b}}^{-\frac{1}{2}}\mathbf{B}\mathbf{D}_{\mathrm{u}}^{-\frac{1}{2}}$
Compute the left singular vectors of $\tilde{\mathbf{B}}$ corresponding to the K largest singular values, represented by $\tilde{\mathbf{b}}_m$, and stack them into the matrices $\tilde{\mathbf{U}}_K$
Treat each row of $\mathbf{U}_K = \mathbf{D}_{\mathrm{b}}^{-\frac{1}{2}}\tilde{\mathbf{U}}_K$ as coordinates and apply K-means to obtain the clusters $\{\mathcal{B}_m\}_{m=1,...,K}$
for *each user* j *in* $\mathcal{U}$ **do**
    Solve the problem (4.1.21)
    Label the user with the label of the optimal cluster
end

4.2 Graph Partitioning defined by Modified-MinMaxCut

4.2.1 Graph Setting

A brief description is made here as recollection. The problem that is considered here is

$$\min_{\{\mathcal{C}_m\}} \; J_2(\{\mathcal{C}_m\}) \tag{4.2.1}$$

$$= \min_{\{\mathcal{C}_m\}} \sum_{m=1}^{K} \frac{\sum_{i\in\mathcal{C}_m, j\in\mathcal{V}\setminus\mathcal{C}_m} w_{i,j}}{\sum_{i\in\mathcal{B}_m, j\in\mathcal{U}_m} w_{i,j}} \tag{4.2.2}$$

$$= \min_{\{\mathcal{C}_m\}} \sum_{m=1}^{K} \frac{cut(\mathcal{C}_m)}{W(\mathcal{B}_m,\mathcal{U}_m)}, \tag{4.2.3}$$

where $\mathcal{B}_m = \mathcal{C}_m \cap \mathcal{B}(\neq \varnothing)$, $\mathcal{U}_m = \mathcal{C}_m \cap \mathcal{U}(\neq \varnothing)$, $\mathcal{C}_1 \cup \mathcal{C}_2 \cup ... \cup \mathcal{C}_K = \mathcal{V}$, $|\mathcal{V}| = N$, and $\mathcal{C}_i \cap \mathcal{C}_j = \varnothing,\ \forall i,j \in \{1,...,K\}, i \neq j$.

Instead of directly addressing the general clustering problem (4.2.1), we first focus on a simpler problem, specifically, the *graph bisection problem*, i.e, $K = 2$. We hope that similarly to the traditional graph partition problem relaxation, the graph bisection problem sheds some light on how to address the multiway partition problem.

4.2.2 Graph Bisection

To address the graph bisection case, we first introduce some required notation and express the cost function of the problem in terms of a sum of ratios of quadratic forms.

4.2.2.1 Cost Function as Sum of Quadratic Ratios

First, we define the following notations as before

- $\mathbf{c}_m \in \{0,1\}^N$: indicator vector for set $\mathcal{C}_m$, and $\sum_{m=1}^{K} \mathbf{c}_m = \mathbf{1}$,
- $\mathbf{W} \in \mathbb{R}^{N\times N}$: the weighted adjacency matrix,
- $\mathbf{L} \in \mathbb{R}^{N\times N}$: the Laplacian matrix,
- $\mathbf{B}_{\text{in}} := \text{diag}(\mathbf{b}) \in \{0,1\}^{N\times N}$: indicator matrix for set $\mathcal{B}$, i.e., $[\mathbf{b}]_i = 1$ if $i \in \mathcal{B}$,
- $\mathbf{U}_{\text{in}} := \text{diag}(\mathbf{u}) \in \{0,1\}^{N\times N}$: indicator matrix for set $\mathcal{U}$, i.e., $[\mathbf{u}]_i = 1$ if $i \in \mathcal{U}$.

Using this notation, we can write the expressions related to the cost function (3.2.3) in matrix-vector form, i.e.,

$$\sum_{i\in\mathcal{C}_m, j\in\mathcal{V}\setminus\mathcal{C}_m} w_{i,j} = \mathbf{c}_m^\top \mathbf{W}(\mathbf{1}-\mathbf{c}_m) = \mathbf{c}_m^\top \mathbf{L}\mathbf{c}_m \tag{4.2.4}$$

and

$$\sum_{i\in\mathcal{B}_m, j\in\mathcal{U}_m} w_{i,j} = \mathbf{c}_m^\top \mathbf{B}_{\text{in}}\mathbf{W}\mathbf{U}_{\text{in}}\mathbf{c}_m = \mathbf{c}_m^\top \tilde{\mathbf{W}}\mathbf{c}_m, \tag{4.2.5}$$

where we have defined $\tilde{\mathbf{W}} := \mathbf{B}_{\text{in}}\mathbf{W}\mathbf{U}_{\text{in}}$. Thus, the cost function (3.2.3) can be rewritten using quadratic forms as

$$J_2(\{\mathbf{c}_m\}) = \sum_{m=1}^{K} \frac{\mathbf{c}_m^\top \mathbf{L}\mathbf{c}_m}{\mathbf{c}_m^\top \tilde{\mathbf{W}}\mathbf{c}_m}. \tag{4.2.6}$$

4.2.2.2 Two-Clusters Cost Function

We now consider the case of graph bisection and find the explicit expression for (4.2.6) when $K = 2$. Note that as there are only two sets, one indicator vector is the direct complement of the other, i.e.,

$$\mathbf{c}_2 = \mathbf{1} - \mathbf{c}_1.$$

Hence, it is sufficient to consider a single variable, we keep it as $\mathbf{c}$. Using this observation, the cost can be given as

$$J_2(\mathbf{c}) = \frac{\mathbf{c}^\top \mathbf{L}\mathbf{c}}{\mathbf{c}^\top \tilde{\mathbf{W}}\mathbf{c}} + \frac{(\mathbf{1}-\mathbf{c})^\top \mathbf{L}(\mathbf{1}-\mathbf{c})}{(\mathbf{1}-\mathbf{c})^\top \tilde{\mathbf{W}}(\mathbf{1}-\mathbf{c})} \tag{4.2.7}$$

$$= \frac{\mathbf{c}^\top \mathbf{L}\mathbf{c}}{\mathbf{c}^\top \tilde{\mathbf{W}}\mathbf{c}} + \frac{\mathbf{c}^\top \mathbf{L}\mathbf{c}}{\kappa - \mathbf{q}^\top\mathbf{c} + \mathbf{c}^\top \tilde{\mathbf{W}}\mathbf{c}}, \tag{4.2.8}$$

where $\kappa := \mathbf{1}^\top \tilde{\mathbf{W}}\mathbf{1}$ and $\mathbf{q} := (\tilde{\mathbf{W}}^\top + \tilde{\mathbf{W}})\mathbf{1}$.

4.2.3 Relaxation: Method 1

At this stage, in a typical graph partition problem, the second term is disregarded (there is no restriction on the type of nodes) and only the first term is considered in the optimization problem. Identifying the first ratio as a *Rayleigh quotient*, a(n) (approximate) solution can be found by solving the related GEVP. However, due to the presence of the linear term, $\mathbf{q}^\top \mathbf{c}$, in (4.2.8), it would be more complicated to obtain the related GEVP. In the following, we first discuss a slightly different formulation to develop the tools to find a solution to the graph bisection problem.

4.2.3.1 Sum of Inverse Ratios

Instead of considering the original problem formulation, let us assume that we want to *maximize* the following cost

$$\tilde{J}_2(\mathbf{c}) := \frac{\mathbf{c}^\top \tilde{\mathbf{W}} \mathbf{c}}{\mathbf{c}^\top \mathbf{L} \mathbf{c}} + \frac{\mathbf{c}^\top \tilde{\mathbf{W}} \mathbf{c} - \mathbf{q}^\top \mathbf{c} + \kappa}{\mathbf{c}^\top \mathbf{L} \mathbf{c}} \tag{4.2.9}$$

$$= \frac{2\mathbf{c}^\top \tilde{\mathbf{W}} \mathbf{c} - \mathbf{q}^\top \mathbf{c} + \kappa}{\mathbf{c}^\top \mathbf{L} \mathbf{c}}. \tag{4.2.10}$$

Here, we have considered the sum of the inverse ratios from (4.2.8). Notice that by considering this change in the cost function, both terms have the same denominator thus can be directly added. Clearly, this cost function is scale sensitive in $\mathbf{c}$ because of the existence of the linear term. If we scale $\mathbf{c}$ with an arbitrarily small scalar, then this cost function would attain positive infinity for any $\mathbf{c}$. Furthermore, remember that we should avoid the solutions $\mathbf{c} = \mathbf{0}$ or $\mathbf{1}$. To solve these issues, as well as to simplify the optimization problem, we adopt the constraint $\mathbf{c}^\top \mathbf{L} \mathbf{c} = 1$ on $\mathbf{c}$. Notice that this constraint holds for both indicator vectors since $(\mathbf{1} - \mathbf{c})^\top \mathbf{L} (\mathbf{1} - \mathbf{c}) = \mathbf{c}^\top \mathbf{L} \mathbf{c} = 1$. Similar constraint expressed by the quadratic form of $\mathbf{c}$ can be seen in [1, 6] as well.

Finally, dropping the binary constraints on $\mathbf{c}$, we are going to solve the optimization problem

$$\max_{\mathbf{c} \in \mathbb{R}^N, \mathbf{c}^\top \mathbf{L} \mathbf{c} = 1} \tilde{J}_2(\mathbf{c}). \tag{4.2.11}$$

Proposition 6. Consider the function $\tilde{J}_2(\mathbf{c})$ in (4.2.11). Its critical points correspond to vectors $\mathbf{c}$ is given by

$$\mathbf{c} = \mathbf{u} + 0.5\mathbf{1},$$

where $\mathbf{u}$ vectors are the generalized eigenvectors satisfying

$$\mathbf{u}^\top \mathbf{W}_\mathrm{S} \mathbf{u} = \lambda, \mathbf{u}^\top \mathbf{L} \mathbf{u} = 1,$$

where $\mathbf{W}_\mathrm{S} := 2(\tilde{\mathbf{W}}^\top + \tilde{\mathbf{W}})$. λ is the generalized eigenvalue related to the generalized eigenvector $\mathbf{u}$.

Proof. Considering the constraint, the Lagrangian is

$$\mathcal{L}(\mathbf{c}, \nu) = \tilde{J}_2(\mathbf{c}) + \nu(\mathbf{c}^\top \mathbf{L} \mathbf{c} - 1).$$

The critical points of the function can be found by taking the gradient of the Lagrangian *w.r.t* $\mathbf{c}$, and we have the following expression

$$\frac{\partial \mathcal{L}}{\partial \mathbf{c}} = \frac{\big[2(\tilde{\mathbf{W}}^\top + \tilde{\mathbf{W}})\mathbf{c} - \mathbf{q}\big](\mathbf{c}^\top \mathbf{L}\mathbf{c}) - 2\mathbf{L}\mathbf{c}(2\mathbf{c}^\top \tilde{\mathbf{W}}\mathbf{c} - \mathbf{q}^\top \mathbf{c} + \kappa)}{(\mathbf{c}^\top \mathbf{L}\mathbf{c})^2} + 2\nu\mathbf{L}\mathbf{c}. \quad (4.2.12)$$

Equating to zero the above equation, and rearranging its terms, we obtain the expression

$$2(\tilde{\mathbf{W}}^\top + \tilde{\mathbf{W}})\mathbf{c} - \mathbf{q} = \frac{2\mathbf{c}^\top \tilde{\mathbf{W}}\mathbf{c} - \mathbf{q}^\top \mathbf{c} + \kappa - \nu(\mathbf{c}^\top \mathbf{L}\mathbf{c})^2}{\mathbf{c}^\top \mathbf{L}\mathbf{c}} 2\mathbf{L}\mathbf{c},$$

which leads to

$$\mathbf{W}_{\mathrm{S}}\mathbf{c} = \lambda \mathbf{L}\mathbf{c} + \mathbf{q}, \quad (4.2.13)$$

where $\mathbf{W}_{\mathrm{S}}$ is defined as before, $\mathbf{q} := (\tilde{\mathbf{W}}^\top + \tilde{\mathbf{W}})\mathbf{1}$, $\lambda := 2(2\mathbf{c}^\top \tilde{\mathbf{W}}\mathbf{c} - \mathbf{q}^\top \mathbf{c} - \nu(\mathbf{c}^\top \mathbf{L}\mathbf{c})^2)(\mathbf{c}^\top \mathbf{L}\mathbf{c})^{-1}$ and $\nu \in \mathbb{R}$ is the Lagrange multiplier. Taking the structure of $\mathbf{q}$ into consideration, we can further simplify (4.2.13) by first rewriting (4.2.13) as

$$2(\tilde{\mathbf{W}}^\top + \tilde{\mathbf{W}})\mathbf{c} = \lambda \mathbf{L}\mathbf{c} + (\tilde{\mathbf{W}}^\top + \tilde{\mathbf{W}})\mathbf{1},$$

and then achieving

$$2(\tilde{\mathbf{W}}^\top + \tilde{\mathbf{W}})(\mathbf{c} - 0.5\mathbf{1}) = \lambda \mathbf{L}\mathbf{c}.$$

Since $\mathbf{1}$ is in the $ker(\mathbf{L})$, we then have

$$\mathbf{W}_{\mathrm{S}}(\mathbf{c} - 0.5\mathbf{1}) = \lambda \mathbf{L}(\mathbf{c} - 0.5\mathbf{1}), \quad (4.2.14)$$

which is a generalized eigenvalue problem (GEVP). And if we denote the generalized eigenvector of (4.2.14) as $\mathbf{u}$ satisfying

$$\mathbf{u}^\top \mathbf{W}_{\mathrm{S}}\mathbf{u} = \lambda, \mathbf{u}^\top \mathbf{L}\mathbf{u} = 1,$$

then we can find that $\mathbf{c} = \mathbf{u} + 0.5\mathbf{1}$.

But in order to show vectors $\mathbf{c}$ are valid critical points of (4.2.11), we have to demonstrate (i) $\mathbf{c}$ should satisfy $\mathbf{c}^\top \mathbf{L}\mathbf{c} = 1$, which is guaranteed by $\mathbf{u}^\top \mathbf{L}\mathbf{u} = 1$ as $\mathbf{1}$ is in the $ker(\mathbf{L})$, and (ii) $\mathbf{c}$ should lead to the equality of λ and the generalized eigenvalue. This can be ensured by having $\nu = 0.5\kappa$. If we equate

$$\frac{\mathbf{u}^\top \mathbf{W}_{\mathrm{S}}\mathbf{u}}{\mathbf{u}^\top \mathbf{L}\mathbf{u}} = 2\frac{2\mathbf{c}^\top \tilde{\mathbf{W}}\mathbf{c} - \mathbf{q}^\top \mathbf{c} + \kappa - \nu(\mathbf{c}^\top \mathbf{L}\mathbf{c})^2}{\mathbf{c}^\top \mathbf{L}\mathbf{c}},$$

and plug in $\mathbf{c} = \mathbf{u} + 0.5\mathbf{1}$, by some manipulations, the fact that $\nu = 0.5\kappa$ can be obtained.

To conclude, vectors $\mathbf{c}$ are critical points. □

So far, how to obtain generalized eigenvectors of (4.2.14) has been provided. But there are a set of vectors that satisfy (4.2.14) and each of them would produce a critical point $\mathbf{c}$ of $\tilde{J}_2$. Therefore, a natural question is which one of these

generalized eigenvectors should be chosen as the continuous indicator vector. In order to determine the solution to the sum of inverse ratios, we first look into

$$\lambda = 2\frac{2\mathbf{c}^\top\tilde{\mathbf{W}}\mathbf{c} - \mathbf{q}^\top\mathbf{c} + \kappa - \nu(\mathbf{c}^\top\mathbf{L}\mathbf{c})^2}{\mathbf{c}^\top\mathbf{L}\mathbf{c}} \tag{4.2.15}$$

$$= 2\tilde{J}_2(\mathbf{c}) - \kappa. \tag{4.2.16}$$

Given the constraint, λ can be identified as the sum of inverse ratios with an extra constant term. Consequently, if $\tilde{J}_2$ is to be maximized, then the continuous indicator vector should be the generalized eigenvector corresponding to the largest eigenvalue, or *the leading generalized eigenvector*.

Similarly to the spectral bisection method for graph partition, a binary (two-cluster) solution, $\mathbf{c}^* \in \{0,1\}^N$, can be retrieved by thresholding $\mathbf{c}$. This thresholding process can be viewed as an 1D K-means and achieved by doing a function value sweep to find the threshold, τ^*, that provides the best function evaluation, i.e.,

$$[\mathbf{c}^*]_i := [\mathbf{c}_{\tau^*}]_i = \begin{cases} 1 & [\mathbf{c}(\lambda^*)]_i \geq \tau^* \\ 0 & [\mathbf{c}(\lambda^*)]_i < \tau^* \end{cases} \; : \; \tau^* = \max_{\tau\in\mathbb{R}_+} \tilde{J}_2(\mathbf{c}_\tau). \tag{4.2.17}$$

Thus, if (4.2.11) wants to be solved, we need to pick the leading generalized eigenvector and threshold it using τ^* such that it maximizes $\tilde{J}_2(\cdot)$. Here we would like to clarify that although the generalized eigenvector is not the same as its related critical point $\mathbf{c}$ due to the constant vector, since we perform a threshold sweep operation, they will produce the same clusters. Therefore, we do not clearly differentiate the roles of the generalized eigenvector and the continuous indicator vector.

After presenting a solution for the alternative cost function, i.e., sum of inverse ratios, in the following, we go back to the original formulation of the sum of ratios to study its critical points and therefore characterizing its solution with the relaxation of abandoning discreteness and change it to a continuous variable.

4.2.3.2 Sum of Ratios

Recall the sum of ratios, for the case $K = 2$, i.e.,

$$J_2(\mathbf{c}) = \frac{\mathbf{c}^\top\mathbf{L}\mathbf{c}}{\mathbf{c}^\top\tilde{\mathbf{W}}\mathbf{c}} + \frac{\mathbf{c}^\top\mathbf{L}\mathbf{c}}{\kappa - \mathbf{q}^\top\mathbf{c} + \mathbf{c}^\top\tilde{\mathbf{W}}\mathbf{c}}. \tag{4.2.18}$$

As we discussed before, we adopt $\mathbf{c}^\top\mathbf{L}\mathbf{c} = 1$. The critical points of (4.2.18) given all the constraints are provided in the following proposition.

Proposition 7. Consider the function $J_2(\mathbf{c})$ as in (4.2.18). Its critical points correspond to vectors $\mathbf{c}$ satisfying the expression

$$\mathbf{W}_\mathrm{S}(\mathbf{c} - 0.5\gamma'\mathbf{1}) = \lambda'\mathbf{L}(\mathbf{c} - 0.5\gamma'\mathbf{1}), \tag{4.2.19}$$

where

$$\lambda' = \frac{4b_1b_2(b_1 + b_2 + \nu' b_1 b_2)}{a(b_1^2 + b_2^2)}, \quad \gamma' = \frac{2b_1^2}{b_1^2 + b_2^2},$$

and $a = \mathbf{c}^\top \mathbf{L}\mathbf{c}$, $b_1 = \mathbf{c}^\top \tilde{\mathbf{W}}\mathbf{c}$, $b_2 = \kappa - \mathbf{q}^\top \mathbf{c} + \mathbf{c}^\top \tilde{\mathbf{W}}\mathbf{c}$ and $\nu' = -\kappa(b_1^2 + b_2^2)^{-1}$ is the Lagrange multiplier. $\mathbf{W}_\mathrm{S}, \mathbf{L}$ are defined as before. The vector $\mathbf{c}$ satisfying the above expression is the given by

$$\mathbf{c} = \mathbf{u} + 0.5\gamma'\mathbf{1},$$

where $\mathbf{u}$ is the generalized eigenvector satisfying

$$\mathbf{u}^\top \mathbf{W}_\mathrm{S}\mathbf{u} = \lambda',\ \mathbf{u}^\top \mathbf{L}\mathbf{u} = 1.$$

Proof. See Appendix (B.1) □

In fact, it is not possible to directly compute the $\mathbf{c}$ using (4.2.19). This is because given the generalized eigenvector $\mathbf{u}$ of (4.2.19), to retrieve $\mathbf{c}$ we have to know γ', but to determine γ', the $\mathbf{c}$ is required. Fortunately, to obtain the clusters from the continuous indicator vectors the exact $\mathbf{c}$ is not required. Since $0.5\gamma'\mathbf{1}$ is sorely an offset, after a parameter search for the optimal threshold, the partitioning results will not be affected by different values of γ'.

From the bisection results for the sum of inverse ratios, the leading generalized eigenvector is chosen since the generalized eigenvalue can be identified as the sum of inverse ratios. Heuristically, the result maximizing the $\tilde{J}_2$ would be a *surrogate* solution to minimizing the J_2. Furthermore, an upper bound of the cost J_2 is given by the following proposition

Proposition 8. The sum of ratios J_2 can be upper-bounded by

$$J_2 < \frac{8}{\lambda'}, \tag{4.2.20}$$

Proof. See Appendix (B.2) □

Therefore, by choosing generalized eigenvector corresponding to larger eigenvalue, we tend to reduce the value of the upper bound of J_2, leading to lower cost function value. The leading generalized eigenvector should be thus chosen as the continuous indicator vector for the bisection problem. Although we approach this problem with two different directions, i.e., maximizing the sum of inverse ratios and minimizing the sum of ratios, they lead to the same results. Additionally, to retrieve the binary indicator vector, a discretize operation is required. One can follow the same way described before to determine the threshold.

4.2.3.3 Heuristic for Multi-way Partition

Furthermore, regarding the K-way partitioning problem, similar to the framework of spectral clustering, heuristically K generalized eigenvectors related to the K largest eigenvalues can be used. One can stack these K vectors into a matrix $\mathbf{U}_K$, then treat each row of $\mathbf{U}_K$ as coordinates and apply K-means. Finally, the

label of each node can be obtained. This process is summarized as the following algorithm.

Algorithm 6: Modified-MinMaxCut Multiway Partition - Method 1

Data: The graph $\mathcal{G}(\mathcal{B},\mathcal{U},\mathcal{E})$, the number of clusters K
Result: Desired K Clusters
1 Construct $\mathbf{W}_{\mathrm{S}} = 2(\mathbf{B}_{\mathrm{in}}\mathbf{W}\mathbf{U}_{\mathrm{in}})^{\top} + 2(\mathbf{B}_{\mathrm{in}}\mathbf{W}\mathbf{U}_{\mathrm{in}})$
2 Compute K generalized eigenvectors corresponds to K largest eigenvalues of $\mathbf{W}_{\mathrm{S}}\mathbf{c} = \lambda\mathbf{L}\mathbf{c}$, denoted by $\mathbf{U}_K$
3 Apply K-means over $\mathbf{U}_K$

4.2.4 Relaxation: Method 2

4.2.4.1 Sum of Inverse Ratios

As we have seen before, there exists a linear term $\mathbf{q}$ in the sum of inverse ratios and ratios. Here, we put forward a relaxation with the same constraint that can eliminate this linear term in another way and the solution is given by a GEVP as well.

Remark 1. Since the indicator vector $\mathbf{c}$ is defined as a binary vector, it can be expressed in a quadratic form as in the following equation

$$\mathbf{q}^{\top}\mathbf{c} = \mathbf{c}^{\top}\mathrm{diag}(\mathbf{q})\mathbf{c}. \tag{4.2.21}$$

The sum of inverse ratios $\tilde{J}_2(\mathbf{c})$ can be rewritten as

$$\tilde{J}_2(\mathbf{c}) := \frac{\mathbf{c}^{\top}\tilde{\mathbf{W}}\mathbf{c}}{\mathbf{c}^{\top}\mathbf{L}\mathbf{c}} + \frac{\mathbf{c}^{\top}\tilde{\mathbf{W}}\mathbf{c} - \mathbf{c}^{\top}\mathrm{diag}(\mathbf{q})\mathbf{c} + \kappa}{\mathbf{c}^{\top}\mathbf{L}\mathbf{c}} \tag{4.2.22}$$

$$= \frac{\mathbf{c}^{\top}(2\tilde{\mathbf{W}} - \mathrm{diag}(\mathbf{q}))\mathbf{c} + \kappa}{\mathbf{c}^{\top}\mathbf{L}\mathbf{c}} \tag{4.2.23}$$

with the same definition and constraints as before. So far, we obtain a cost function without the linear term. We can identify the critical points of $\tilde{J}_2$ by taking the gradient *w.r.t* $\mathbf{c}$ towards the Lagrangian and set it to $\mathbf{0}$, then we have the following equation

$$\frac{\partial\mathcal{L}}{\partial\mathbf{c}} = \frac{\big[(\mathbf{W}_{\mathrm{D}}^{\top} + \mathbf{W}_{\mathrm{D}})\mathbf{c}\big](\mathbf{c}^{\top}\mathbf{L}\mathbf{c}) - 2\mathbf{L}\mathbf{c}(\mathbf{c}^{\top}\mathbf{W}_{\mathrm{D}}\mathbf{c} + \kappa)}{(\mathbf{c}^{\top}\mathbf{L}\mathbf{c})^2} + 2\nu\mathbf{L}\mathbf{c}. \tag{4.2.24}$$

where $\mathbf{W}_{\mathrm{D}} = 2\tilde{\mathbf{W}} - \mathrm{diag}(\mathbf{q})$.

Equating to zero the above equation, and rearranging its terms, we obtain the expression

$$(\mathbf{W}_{\mathrm{D}}^{\top} + \mathbf{W}_{\mathrm{D}})\mathbf{c} = 2\frac{\mathbf{c}^{\top}\mathbf{W}_{\mathrm{D}}\mathbf{c} + \kappa - \nu(\mathbf{c}^{\top}\mathbf{L}\mathbf{c})^2}{\mathbf{c}^{\top}\mathbf{L}\mathbf{c}}\mathbf{L}\mathbf{c}, \tag{4.2.25}$$

leading to

$$\mathbf{W}_{\mathrm{DS}}\mathbf{c} = \lambda\mathbf{L}\mathbf{c}, \tag{4.2.26}$$

where $\mathbf{W}_{\mathrm{DS}} = (\mathbf{W}_{\mathrm{D}}^{\top} + \mathbf{W}_{\mathrm{D}})$, and $\nu = \kappa$. And λ can be identified as $\tilde{J}_2(\mathbf{c})$ with an extra constant value. Thus finding the solution for (4.2.11) (without restrictions on $\mathbf{c}$, i.e., $\mathbf{c} \in \mathbb{R}^N$) is equivalent to finding the maximum λ that satisfies (4.2.26) and meanwhile is equal to the corresponding $\tilde{J}_2(\mathbf{c})$ determined by the solution $\mathbf{c}$, which is naturally satisfied. Notice that the problem itself is a GEVP since we transform the linear term into a quadratic form. The optimal $\mathbf{c}$ is the one generating the lowest sum of ratios value.

In order to obtain the label for each node, we should make the (continuous) eigenvector to be binary by setting a threshold. We can thus do a linear search over the threshold to find the best τ^*. The solution should be the thresholded eigenvector generating the lowest sum of ratios value.

4.2.4.2 Sum of Ratios

Recall the sum of ratios, for the case $K = 2$, i.e.,

$$J_2(\mathbf{c}) = \frac{\mathbf{c}^{\top}\mathbf{L}\mathbf{c}}{\mathbf{c}^{\top}\tilde{\mathbf{W}}\mathbf{c}} + \frac{\mathbf{c}^{\top}\mathbf{L}\mathbf{c}}{\kappa - \mathbf{q}^{\top}\mathbf{c} + \mathbf{c}^{\top}\tilde{\mathbf{W}}\mathbf{c}} \tag{4.2.27}$$

$$= \frac{\mathbf{c}^{\top}\mathbf{L}\mathbf{c}}{\mathbf{c}^{\top}\tilde{\mathbf{W}}\mathbf{c}} + \frac{\mathbf{c}^{\top}\mathbf{L}\mathbf{c}}{\kappa - \mathbf{c}^{\top}\mathrm{diag}(\mathbf{q})\mathbf{c} + \mathbf{c}^{\top}\tilde{\mathbf{W}}\mathbf{c}}. \tag{4.2.28}$$

The critical points for (4.2.28) are given in the following proposition.

Proposition 9. Consider the function $J_2(\mathbf{c})$ as in (4.2.28). Its critical points correspond to vectors $\mathbf{c}$ satisfying the expression

$$\mathbf{L}\mathbf{c} = J_2(\mathbf{c})\mathbf{W}_{\mathrm{W}}\mathbf{c}, \tag{4.2.29}$$

where $\mathbf{L}$ is defined as in (4.2.13), and

$$\mathbf{W}_{\mathrm{W}} = \alpha(\tilde{\mathbf{W}}^{\top} + \tilde{\mathbf{W}}) - \theta\mathrm{diag}(\mathbf{q}),$$

$$\alpha = \frac{b_1^2 + b_2^2}{2(b_1 + b_2)(b_1 + b_2 + b_1 b_2 \nu)}, \quad \theta = \frac{a b_1}{b_2(b_1 + b_2 + b_1 b_2 \nu)},$$

where $b_1 = \mathbf{c}^{\top}\tilde{\mathbf{W}}\mathbf{c}$, $b_2 = \kappa - \mathbf{c}^{\top}\mathrm{diag}(\mathbf{q})\mathbf{c} + \mathbf{c}^{\top}\tilde{\mathbf{W}}\mathbf{c}$, and ν is the Lagrange multiplier.

Proof. See Appendix (B.1) □

However, it is impossible to solve this problem directly since the matrix $\mathbf{W}_{\mathrm{W}}$ itself is entangled with $\mathbf{c}$. If we would like to derive the $\mathbf{c}$, we should determine $\mathbf{W}_{\mathrm{W}}$ first but the determination of $\mathbf{W}_{\mathrm{W}}$ depends on the value of $\mathbf{c}$. But there exists an iterative way to determine (one of the) critical points of this GEVP. We notice that:

$$\mathbf{c} = J_2(\mathbf{c})\mathbf{L}^{\dagger}\mathbf{W}_{\mathrm{W}}\mathbf{c}, \tag{4.2.30}$$

where $\mathbf{L}^{\dagger}$ is the pseudo inverse since the Laplacian matrix is rank deficient. We can follow the fixed point iteration, shown as follows:

$$\mathbf{c}^{(n+1)} = J_2(\mathbf{c}^{(n)})\mathbf{L}^{\dagger}\mathbf{W}_{\mathrm{W}}^{(n)}\mathbf{c}^{(n)}, \tag{4.2.31}$$

and finally it will converge to one of the critical points, though the optimality is not guaranteed.

Based on the discussions in the previous sections, the multiway partition solution related to this second relaxation method is provided in Algorithm 7.

Algorithm 7: Modified-MinMaxCut Multiway Partition - Method 2

Data: The graph $\mathcal{G}(\mathcal{B},\mathcal{U},\mathcal{E})$, the number of clusters K
Result: Desired K Clusters
1 Construct $\mathbf{W}_{\mathrm{DS}} = (2\mathbf{B}_{\mathrm{in}}\mathbf{W}\mathbf{U}_{\mathrm{in}} - \mathrm{diag}(\mathbf{q}))^{\top} + (2\mathbf{B}_{\mathrm{in}}\mathbf{W}\mathbf{U}_{\mathrm{in}} - \mathrm{diag}(\mathbf{q}))$
2 Compute K generalized eigenvectors generating K smallest cost function values, denoted by $\mathbf{U}_K$
3 Apply K-means over $\mathbf{U}_K$

4.2.5 Summary

Here, we would like to mention again that the continuous indicator vector is not exactly the same as the generalized eigenvector. But since after discretization operation the clustering results will not be affected, for the sake of notation simplification, we do not show the complicated relationship between the generalized eigenvector and the continuous indicator vector but to provide an illustrative conclusion. The first method is directly derived from the cost function and the inverse cost function, and based on the GEVP (4.2.32). In the relaxation, we drop the binary constraint and adopt the quadratic constraint on the indicator vector $\mathbf{c}$. We first take the derivative towards the indicator vector $\mathbf{c}$ and then set the gradient to $\mathbf{0}$. This leads us to

$$\mathbf{W}_{\mathrm{S}}\mathbf{c} = \lambda\mathbf{L}\mathbf{c}, \tag{4.2.32}$$

and consequently, the continuous indicator vector is the generalized eigenvector of this GEVP. What's more, we relate the eigenvalue to the cost function value, indicating that the leading generalized eigenvector should be chosen as the indicator vector for the bisection problem. Following the multi-way partition method in spectral clustering, we can pick the eigenvectors related to the K largest eigenvalues and stack them into the matrix $\mathbf{U}_K$. Considering each row as the spectral embedding of each node, any clustering algorithm can be applied, e.g., K-means, to obtain the labels of nodes.

$$\mathbf{W}_{\mathrm{DS}}\mathbf{c} = \lambda\mathbf{L}\mathbf{c} \tag{4.2.33}$$

The second method is based on the GEVP (4.2.33). In order to relax the original problem and obtain this GEVP, here we adopt a transformation. Given that the final solution is binary, the linear term can be rewritten as a quadratic term and be absorbed. Then we can get rid of this linear term and turn the equation to a GEVP. All the critical points are given by the eigenvectors of this GEVP and the one which minimizes the cost function is the optimal (continuous) indicator vector for the bisection. As before, the multiway partition solution can be extended by taking more eigenvectors in to consideration.

Bibliography

[1] F. Nie, C. Ding, D. Luo, and H. Huang, "Improved minmax cut graph clustering with nonnegative relaxation," in *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*. Springer, 2010, pp. 451–466.

[2] M. Gu, H. Zha, C. Ding, X. He, H. Simon, and J. Xia, "Spectral relaxation models and structure analysis for k-way graph clustering and bi-clustering," 2001.

[3] I. S. Dhillon, "Co-clustering documents and words using bipartite spectral graph partitioning," in *Proceedings of the seventh ACM SIGKDD international conference on Knowledge discovery and data mining*, 2001, pp. 269–274.

[4] C. H. Ding, X. He, H. Zha, M. Gu, and H. D. Simon, "A min-max cut algorithm for graph partitioning and data clustering," in *Proceedings 2001 IEEE international conference on data mining*. IEEE, 2001, pp. 107–114.

[5] H. Zha, X. He, C. Ding, H. Simon, and M. Gu, "Bipartite graph partitioning and data clustering," in *Proceedings of the tenth international conference on Information and knowledge management*, 2001, pp. 25–32.

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

Case Study: Simple Static Graph for Methods Validation and Comparison for Problem 2

5

In this Chapter, we simulate the bisection approaches and multiway heuristic methods introduced in Chapter 4 for the second problem on a randomly generated small-size graph. We compare different relaxation ways and discuss their performance.

5.1 Practicalities

To obtain the (continuous) indicator vectors, we have to solve GEVPs and computing generalized eigenvectors as

$$\mathbf{U} = \mathbf{U}_{\mathbf{L}}\mathbf{\Sigma}_{\mathbf{L}}^{-\frac{1}{2}}\mathbf{U}_{\tilde{\mathbf{A}}} \tag{5.1.1}$$

where we have

$$\tilde{\mathbf{A}} = (\mathbf{U}_{\mathbf{L}}\mathbf{\Sigma}_{\mathbf{L}}^{-\frac{1}{2}})^{\top}\mathbf{W}_{\mathrm{S}}(\mathbf{U}_{\mathbf{L}}\mathbf{\Sigma}_{\mathbf{L}}^{-\frac{1}{2}}). \tag{5.1.2}$$

However, in practice, the matrices $\mathbf{L}$ and $\mathbf{W}_{\mathrm{S}}$ (or $\mathbf{W}_{\mathrm{DS}}$) cannot be expected to be full-rank. In fact, by mere definition, $\mathbf{L}$ is not, i.e., it has (at least) the constant vectors in its null space. And usually in practice, $\mathbf{W}_{\mathrm{S}}$ (or $\mathbf{W}_{\mathrm{DS}}$) is rank deficient as well. Thus, computing generalized eigenvectors as (5.1.1) and (5.1.2) is not possible. Notice that in this case $\mathbf{\Sigma}_{\mathbf{L}}$ is not invertible, i.e., $\mathbf{L}$ is not full-rank. Hence, instead of using $\mathbf{\Sigma}_{\mathbf{L}}$, we can reshape it to an $(N-1)\times(N-1)$ matrix by eliminating the zero eigenvalue. After that, we can follow the same procedure to construct $\mathbf{U}$. But note that in this case, the $\mathbf{U}$ is an tall, $N\times(N-1)$ matrix.

In the following sections, we will consider an instance of a community graph with $N = 40$ nodes [1], shown in Fig. 5.1. We will partition this graph using the proposed relaxation methods of the *Modified-MinMaxCut* and explain the implementation of relaxation methods using this example. Finally, we will discuss the performance of these methods.

In this example, we consider a binary adjacency matrix $\mathbf{W}$, and $\mathbf{L}$ as its related combinatorial Laplacian. The set $\mathcal{B}$ is defined as a realization of a Bernoulli process with success probability $p = 0.5$, that is, we randomly define the type of each node. The set $\mathcal{U}$ is defined as its complement, i.e., $\mathcal{B}\cup\mathcal{U} = \mathcal{V}$. As $\mathbf{L}$ is rank-deficient, we first construct $\mathbf{U}$ by computing the eigenvectors of $\mathbf{L}$ and $\tilde{\mathbf{A}}$ related to eigenvalues larger than eps $= 10^{-6}$.

5.2 Simulation: Method 1

The solution to the bisection problem can be obtained by solving the GEVP (4.2.32) and the continuous indicator vector is simply the generalized eigenvector corresponding to the largest eigenvalue. Furthermore, the bisection solution

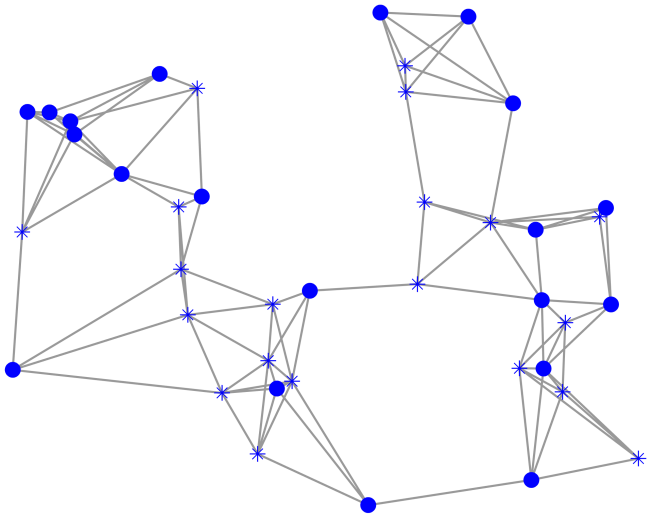


Figure 5.1: Graph example with two types of nodes, denoted by circles and stars.

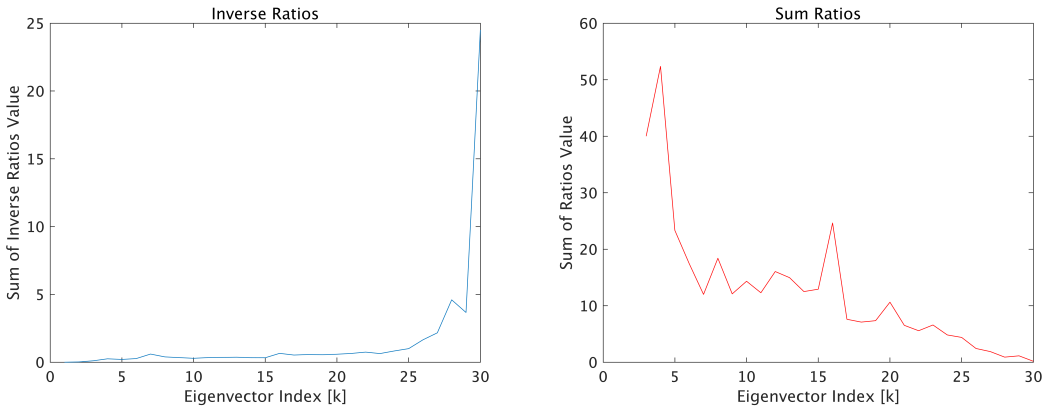


Figure 5.2: Comparison of function value for different generalized eigenvectors in method 1. Left: Sum of Inverse Ratios. Right: Sum of Ratios. (eigenvalues from small to large)

can be extended to multiway partition problem based on spectral clustering framework. We will first illustrate graph bisection results and then go to the multiway simulation results.

We show the results by comparing the function value obtained by the discretized columns of $\mathbf{U}$, for both sums of ratios and inverse ratios. The results are shown in Fig. 5.2. In these plots, it is seen that indeed, the leading generalized eigenvector achieving the highest function evaluation in the inverse ratio cost function achieves the minimum in the sum of ratios.

Now, we illustrate the partitioning results, including the bisection and the 3-way partition in Fig. 5.3. The left one is obtained by thresholding the leading generalized eigenvectors while the right one is generated by the leading K eigenvectors and K-means, where $K = 3$. And these partitions generate the following

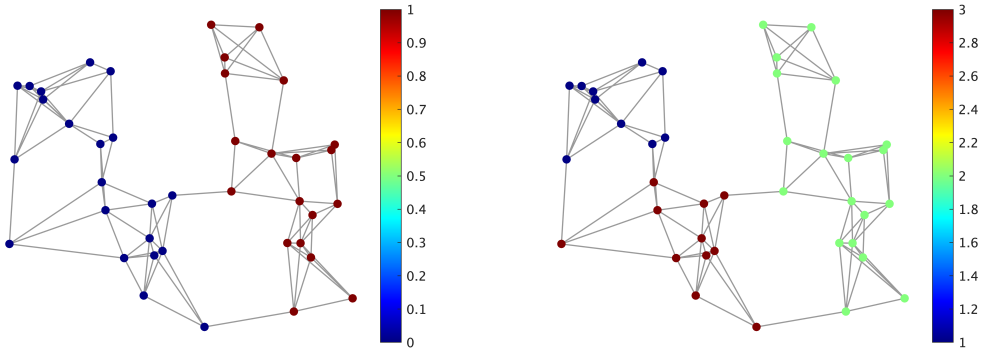


Figure 5.3: Method 1 solutions. Left: Bisection. Right: 3-way partition.

cost values, which validate the performance of the spectral clustering-wise method.

```
Bisection Cost: 0.16333
Multiway partition Cost: 0.98000
```

5.3 Simulation: Method 2

Method 2 is related to the GEVP (4.2.26), which is derived from the inverse cost function. The bisection solution is obtained by the generalized eigenvector which minimizes the cost function. In order to show that, the $\tilde{J}_2$ and J_2 value regarding each eigenvector is given in Fig. 5.4.

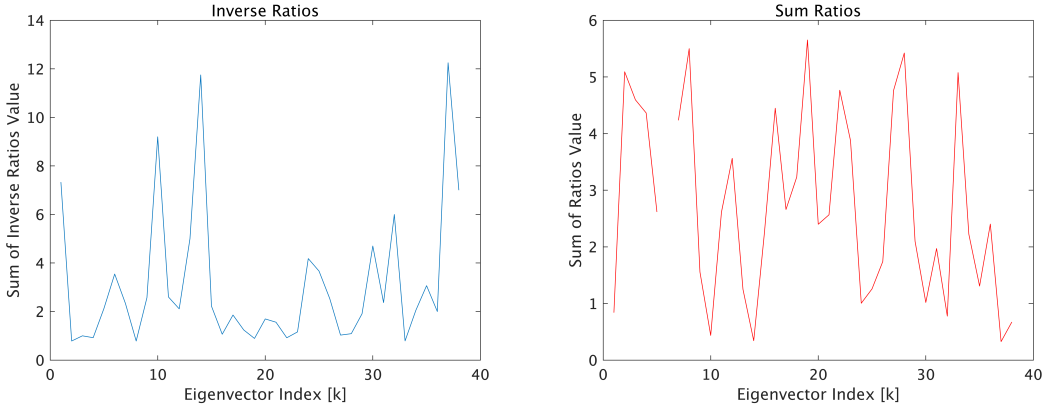


Figure 5.4: Comparison of function value for different generalized eigenvectors in method 2. Left: Sum of Inverse Ratios. Right: Sum of Ratios. (eigenvalues from small to large)

In Fig. 5.5, we give the results for bisection and multiway partition. One can observe that even for such a simple small size graph, the bisection solution has a minor difference compared with the results in the previous section. For the multiway partition, this method behaves even worse, especially on the boundary of the second cluster and the third cluster. The cost function values are given as follows, where large degenerations can be observed.

```
Bisection Cost: 0.32667
Multiway partition Cost: 2.34520
```

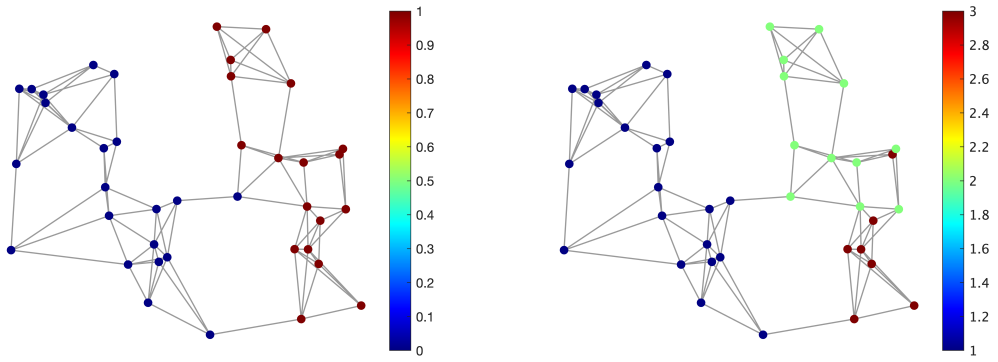


Figure 5.5: Method 2 solutions. Left: Bisection. Right: 3-way partition.

5.4 Discussions

So far, we have implemented all the bisection and multiway partition solutions over a small-size static graph. In the following part, we are going to discuss the advantages and drawbacks of each method, and compare these solutions according to their graph partitioning performances.

Let's look into the first method. In the first method, we directly relax the cost function of the bisection problem, then extend the approach to multiway partition using the framework of spectral clustering, where the (continuous) indicator vector is based on the generalized eigenvector of the GEVP (4.2.32). According to the simulation results, this method performs quite well not only in bisection problem, but also in the multiway case. Then let's go to the second method. In the second method, we use a trick to eliminate the linear term in the cost function. However, the generated clusters are worse than the clusters generated by the first problem. This can be attributed to the fact that the transformation is unsatisfactory. Although for a binary vector the transformation in the Remark 1 always holds, yet in the relaxation, we first go to the continuous indicator vectors and then discretize them and the transformation no longer holds for the continuous indicator vectors. Therefore, the error is introduced when we relax the binary constraint. Furthermore, the solution is given by the sum of inverse ratios instead of the original cost function. For the bisection problem, we might obtain good clusters. But for the multiway partition problem, the solution could be worse. This is the main reason for the poor performance in terms of the cost function value. And one may notice that we don't relate the continuous indicator vectors to the leading K generalized eigenvectors of the GEVP (4.2.26) since we found the matrix $\mathbf{W}_{\text{DS}}$ is negative semidefinite, meaning that all the eigenvalues of the GEVP (4.2.26) is not positive and cannot be identified as the value of the sum of inverse ratios. It would be meaningless to select the eigenvector corresponding to the largest eigenvalue as the continuous indicator vector. The proof of negative semidefinite

can be found in Appendix B.3. To sum up, the second relaxation method is not as satisfying as the first relaxation method. Therefore, in the adaptive part, the adaptive solutions are generated *only* using the first method.

Both solutions suffer from the computational inefficiency since the EVD is required to obtain the spectral embeddings of nodes. This could be a severe issue when the graph size becomes larger and adaptively cluster update is needed. To tackle this issue, in the next Chapter, we are going to discuss how to efficiently update the clusters of a graph changing over time and avoid computing EVD every time.

Bibliography

[1] N. Perraudin, J. Paratte, D. Shuman, L. Martin, V. Kalofolias, P. Vandergheynst, and D. K. Hammond, "GSPBOX: A toolbox for signal processing on graphs," *ArXiv e-prints*, Aug. 2014.

6 Adaptive Solutions

As we have discussed in Chapter 3, another aspect of this thesis pertains to derive solutions that can partition graphs changing over time. Minimization strategies for the time-invariant cost functions introduced in Chapter 3 have been introduced in the Chapter 4, in this Chapter, we discuss adaptive strategies for the graph clustering problem.

6.1 Literature Review

When the graph that needs to be partitioned changes over time, a natural question is whether there is a way to update the clusters. For a large graph, the size of its adjacency or Laplacian matrix is large as well. As we have already discussed in Chapter 4, all the continuous indicator vectors towards the graph clustering problem defined by (3.1.3) and (3.2.3) can be computed using GEVPs. Therefore for large graphs, it might be infeasible to compute the generalized eigenvectors from scratch every time an update of the clusters is required. As a consequence, we should derive a computation-efficient way to update the clusters.

In the previous chapter, we derived several solutions to partition graphs under certain cost functions. The solutions include performing recursive bisection and methods that are similar to spectral clustering. For a graph changing over time, it is difficult to update the clusters when using the recursive bisection, since in each update, we have to start the recursive bisection from the beginning, which is expensive. Therefore, we have to turn to other solutions. For example, since the clusters are derived by applying clustering algorithms, e.g., K-means, to a subset of computed eigenvectors, one way to update the clusters is to update the eigenvectors and then updating the previous K-means centroids using the updated eigenvectors. Eigenvectors updating is related to many topics, such as subspace tracking or subspace updating, eigensystem updating, and matrix perturbation theory. This tells us the *adaptive clusters updating problem* can be translated into an *(generalized) eigenvectors updating problem*. Based on this idea, we review several works on this topic. The following three topics are parallel and all related to the update of clusters.

6.1.1 Subspace Updating

Subspace tracking or subspace updating is strongly related to subspace-based applications, such as Direction-Of-Arrival (DOA) estimation using ESPRIT or MUSIC [1]. The basic idea is that the eigenvectors of the (sample) data covariance matrix should be updated when a new signal snapshot is received, such that the information which is potentially slowly varying and carried by the signal snapshots can be extracted. In these applications, one usually works on the sample covari-

ance matrix estimated from noisy signal snapshots. Among all the techniques, one of the most famous is the Projection Approximation Subspace Tracking (PAST) technique [1]. In this technique, the desired subspace is treated as the optimum of an unconstrained optimization problem. In order to reach the optimal point, the gradient descent and recursive least squares techniques are combined and implemented. The PAST-like algorithm is related to the power method as well. In [2], some mathematical proofs are given to clarify their relationship. Apart from the PAST, there are many other methods to track the subspace. To complete the review, many variants of the PAST method to update the subspace are described in [3, 4]. Finally, besides PAST, other ways to translate a desired subspace into the optimum of the unconstrained/constrained optimization problem can be found in [5–9].

Apart from these iterative tracking methods, the incremental SVD update [10–12], which is based on the SVD of a smaller size matrix, is designed to update the SVD of a big fat matrix. There are a few works related to tracking the minor subspace [13–16], or the noise subspace. Additionally, the Kalman filter is implemented as well to track the principal subspace [17].

Despite the fact that these methods have been applied successfully, in other applications, all of them require a gap between the principle subspace (the signal subspace) and the minor subspace components (the noise subspace), or the knowledge of the noise level. Unfortunately, in our case, we do not have such information. Therefore, it is not advisable to directly turn to these methods.

6.1.2 Matrix Perturbation

As we have shown before, generating the partitions is equivalent to solving a GEVP. The variations of the graph can be seen as perturbations on the adjacency matrix or Laplacian matrix. Therefore, updating the eigenvectors (singular vectors) is equal to updating the solutions when the matrices in the GEVP vary. When the variations are small, they can be seen as perturbations of these matrices, leading us to methods based on matrix perturbation theory.

The first approach based on matrix perturbation theory is introduced in [18], and applied to the DOA estimation problem in [19]. In this approach, the update of the eigenvalues is based on the Geršgorin theorem [20]. In order to update the eigenvectors, the perturbations on the eigenvectors are decomposed as the linear combination of all the eigenvectors. Using the orthogonality of the eigencomponents, one can derive the contribution of each eigencomponent to the perturbations of the generalized eigenvector. However, this approach apparently requires all the eigencomponents, which is infeasible when the size of the graph is large. In order to circumvent this problem, Chen et al. [21] propose two methods called **Trip-basic** and **Trip** to simplify the update procedure. In these two methods, not all the components but the first K components are taken into consideration, which, however, makes us lose a lot of information when $K \ll N$. An extension of this method to the GEVP can be found in [22]. Another alternative is introduced in [23]. This approach updates the eigencomponents not by computing the linear combination of eigenvectors, but by a heuristic update. Since the method presented in this thesis is based on this method, we provide details of this method

together with the introduction of ours later in this chapter.

The second approach relying on matrix perturbation theory is based on the invariant subspace. The invariant subspace theory is widely discussed in [24] and applied to the DOA estimation problem in [25]. However, this method requires the information of all the eigencomponents as well, which is infeasible.

6.1.3 Incremental Spectral Clustering

The spectral clustering-based adaptive partition update is called incremental spectral clustering [26], or online spectral clustering. In [27], Christoffer et al. give a general framework for the incremental spectral clustering and discuss its application in topological mapping. Similarly, in [28], the authors derive an algorithm based on incremental spectral clustering for the localization in outdoor environments. In [29], Kong et al. derive an algorithm based on the Ng-Jordan-Weiss (NJW) incremental spectral clustering method [30], which is able to partition a large data set efficiently by compressing the original data set while maintaining the representative points and performing a continuous update of the eigensystem. However, these methods do not address partitioning the graph when edge weights change [31–33]. To deal with this, by incorporating the idea of Laplacian matrix decomposition, Ning et al. [23] represent the similarity variation as new columns to the incidence matrix and then proceed to update the clusters.

6.2 Update Procedure

In this part, we proceed to derive the update procedure for the clusters. We formulate the variations in the graph topology as perturbations to the graph matrices, such as the Laplacian matrix or adjacency matrix. Since in spectral clustering, the partitioning nodes are related to the eigenvectors or the singular vectors of such matrices, the main idea of the introduced update method is that based on the current information of the eigenpairs, we approximate the eigenpairs for the perturbed graph and update accordingly the clusters. Thus, it is necessary to initialize the method with exactly computed eigenpairs, but during the update, there is no need to compute the EVD or SVD from scratch every time.

The derivation of the update procedure is based on the notation of the second graph partitioning problem. But the procedure is exactly the same regarding the first research problem of this thesis work, which will be summarized in the second part. Additionally, we would like to mention that although the generalized eigenvectors are not the critical points of the cost function, due to the discussion that they will lead to the same clustering results, for the sake of notation and to simplify the discussion, we will equate the role of continuous indicator vector and the generalized eigenvector.

6.2.1 Modified-MinMaxCut: Derivation of update

The m-th continuous indicator vectors can be derived from the following GEVP

$$\mathbf{W}_{\mathrm{r}}\mathbf{c}_m = \lambda_m \mathbf{L}\mathbf{c}_m \tag{6.2.1}$$

and its perturbed eigensystem:

$$(\mathbf{W}_\mathrm{r} + \Delta\mathbf{W}_\mathrm{r})(\mathbf{c}_m + \Delta\mathbf{c}_m) = (\lambda_m + \Delta\lambda_m)(\mathbf{L} + \Delta\mathbf{L})(\mathbf{c}_m + \Delta\mathbf{c}_m). \tag{6.2.2}$$

Here, we define matrices $\mathbf{W}_\mathrm{r}$, where subscript r is a place holder, to represent $\mathbf{W}_\mathrm{S}$, or $\mathbf{W}_\mathrm{DS}$, for the sake of convenience. We would like to clarify that $\mathbf{W}_\mathrm{r}$ and $\mathbf{L}$ do not have to be full rank in this case. Note thereby that the Laplacian matrix is not full rank. $\mathbf{c}_m$ is the (continuous) indicator vector for subgraph $\mathcal{C}_m$.

Proposition 10. The $\Delta\lambda_m$ defined in (6.2.2) can be determined by the following equation

$$\Delta\lambda_m = \frac{\mathbf{c}_m^\top(\Delta\mathbf{W}_\mathrm{r} - \lambda_m\Delta\mathbf{L})(\mathbf{c}_m + \Delta\mathbf{c}_m)}{\mathbf{c}_m^\top(\mathbf{L} + \Delta\mathbf{L})(\mathbf{c}_m + \Delta\mathbf{c}_m)}. \tag{6.2.3}$$

Proof. We expand (6.2.2) and obtain

$$\begin{aligned}
&\Delta\mathbf{W}_\mathrm{r}\mathbf{c}_m + \mathbf{W}_\mathrm{r}\Delta\mathbf{c}_m + \Delta\mathbf{W}_\mathrm{r}\Delta\mathbf{c}_m && (6.2.4)\\
=&\Delta\lambda_m\mathbf{L}\mathbf{c}_m + \lambda_m\Delta\mathbf{L}\mathbf{c}_m + \lambda_m\mathbf{L}\Delta\mathbf{c}_m + \Delta\lambda_m\Delta\mathbf{L}\mathbf{c}_m && (6.2.5)\\
+&\lambda_m\Delta\mathbf{L}\Delta\mathbf{c}_m + \Delta\lambda_m\mathbf{L}\Delta\mathbf{c}_m + \Delta\lambda_m\Delta\mathbf{L}\Delta\mathbf{c}_m. && (6.2.6)
\end{aligned}$$

Then we can left multiply (6.2.6) with $\mathbf{c}_m^\top$, using the fact that $\mathbf{c}_m^\top\mathbf{W}_\mathrm{r} = \lambda_m\mathbf{c}_m^\top\mathbf{L}$, we obtain

$$\begin{aligned}
&\mathbf{c}_m^\top\Delta\mathbf{W}_\mathrm{r}\mathbf{c}_m + \mathbf{c}_m^\top\Delta\mathbf{W}_\mathrm{r}\Delta\mathbf{c}_m && (6.2.7)\\
=&\mathbf{c}_m^\top\Delta\lambda_m\mathbf{L}\mathbf{c}_m + \mathbf{c}_m^\top\lambda_m\Delta\mathbf{L}\mathbf{c}_m + \mathbf{c}_m^\top\Delta\lambda_m\Delta\mathbf{L}\mathbf{c}_m && (6.2.8)\\
+&\mathbf{c}_m^\top\lambda_m\Delta\mathbf{L}\Delta\mathbf{c}_m + \mathbf{c}_m^\top\Delta\lambda_m\mathbf{L}\Delta\mathbf{c}_m + \mathbf{c}_m^\top\Delta\lambda_m\Delta\mathbf{L}\Delta\mathbf{c}_m. && (6.2.9)
\end{aligned}$$

With some manipulations, we can obtain the update equation. □

The determination of $\Delta\mathbf{c}_m$ can be derived from the expansion of (6.2.2) as well. We move all the terms that are related to $\Delta\mathbf{c}_m$ to the left hand side and the rest to the right hand side. This way, we obtain

$$\begin{aligned}
&(\mathbf{W}_\mathrm{r} + \Delta\mathbf{W}_\mathrm{r} - (\lambda_m + \Delta\lambda_m)(\mathbf{L} + \Delta\mathbf{L}))\Delta\mathbf{c}_m && (6.2.10)\\
=&(\Delta\lambda_m\mathbf{L} + \lambda_m\Delta\mathbf{L} + \Delta\lambda_m\Delta\mathbf{L} - \Delta\mathbf{W}_\mathrm{r})\mathbf{c}_m && (6.2.11)
\end{aligned}$$

which can be written as $\tilde{\mathbf{K}}\Delta\mathbf{c}_m = \tilde{\mathbf{h}}$.

Since $\tilde{\mathbf{K}}$ is a singular matrix, a unique $\Delta\mathbf{c}_m$ cannot be determined. Two methods to circumvent this obstacle were proposed in [23, 34]. Both of them can solve the problem, though they are not desirable. The first method is based on the assumption that if the weight between i and j changes, only the entries of the eigenvectors that correspond to i, j and their neighbours change. This assumption means $\Delta\mathbf{c}_m$ only has a few non-zero entries. Using this assumption, the authors define $\mathcal{N}_{ij} = \{k | w_{ik} > 0 \ or \ w_{jk} > 0\}$. For $k \notin \mathcal{N}_{ij}$, $[\Delta\mathbf{c}_m]_k = 0$, and thus the corresponding columns in $\tilde{\mathbf{K}}$ can be eliminated. Under these circumstances, $\Delta\mathbf{c}_m$ can be computed as:

$$[\Delta\mathbf{c}_m]_{\mathcal{N}_{ij}} = (\tilde{\mathbf{K}}_{\mathcal{N}_{ij}}^T\tilde{\mathbf{K}}_{\mathcal{N}_{ij}})^{-1}\tilde{\mathbf{K}}_{\mathcal{N}_{ij}}\tilde{\mathbf{h}} \tag{6.2.12}$$

The computation cost can be reduced if i and j only have a few neighbours. The drawback of this method is that it only approximates the $\Delta\mathbf{c}_m$. Issues of the approximation related to the leaf nodes of the graph are discussed in [23].

The second solution proposed in [34] is using $(\mathbf{W}_\mathrm{r} + \Delta\mathbf{W}_\mathrm{r} - \lambda_m\mathbf{L})$ as $\tilde{\mathbf{K}}$ and then inverts the matrix. Although their simulation shows better performance, it is still infeasible to compute the inversion of such a large matrix.

Here, we propose a novel way to determine $\Delta\mathbf{c}_m$. In the methods we discussed, we should ensure that the perturbations of the matrices are small enough such that the approximation of the generalized eigenvectors and eigenvalues hold. However, even if the perturbation is large, one can slice the perturbation such that the perturbation at each update is small enough [22]. This provides us with the intuition that the $\Delta\mathbf{c}_m$ must be small in the l_2-norm sense as well since we don't perturb the system too much. Therefore, we can add the norm of the perturbation as a regularization term when solving the related least squares problem.

First, we have $\tilde{\mathbf{K}}\Delta\mathbf{c}_m = \tilde{\mathbf{h}}$, which each candidate of $\Delta\mathbf{c}_m$ must satisfy. Furthermore, the $\Delta\mathbf{c}_m$ should be small since the perturbation is small. This tells us that a term controlling the norm of $\Delta\mathbf{c}_m$ should be made. As a result, $\Delta\mathbf{c}_m$ can be retrieved by solving the following optimization problem:

$$\min_{\Delta\mathbf{c}_m} ||\tilde{\mathbf{K}}\Delta\mathbf{c}_m - \tilde{\mathbf{h}}||_2^2 + \epsilon||\Delta\mathbf{c}_m||_2^2, \tag{6.2.13}$$

where ϵ is a regularization parameter.

This is the well-known Tikhonov regularization, which has the closed form solution:

$$\Delta\mathbf{c}_m = (\tilde{\mathbf{K}}^\top\tilde{\mathbf{K}} + \epsilon\mathbf{I})^{-1}\tilde{\mathbf{K}}^\top\tilde{\mathbf{h}} \tag{6.2.14}$$

However, to solve (6.2.13) efficiently, we first rephrase it into a linear system:

$$(\tilde{\mathbf{K}}^\top\tilde{\mathbf{K}} + \epsilon\mathbf{I})\Delta\mathbf{c}_m = \tilde{\mathbf{K}}^\top\tilde{\mathbf{h}} \tag{6.2.15}$$

which can then be solved efficiently using the conjugate gradient descent method [35]. The conjugate gradient descent approach is summarized in Algorithm 10 of Appendix A.

So far, we have derived the matrix perturbation-based eigenvector tracking and updating method. To track the first K eigenvectors, we first initialize the method with the eigenpairs computed by the EVD and then apply the method on each of the components and update eigenvectors one by one when the graph changes. However, as we have described in the beginning, our method is incapable of ensuring that the tracked K eigenvectors are the ones related to the K largest eigenvalues. For example, as the perturbations accumulate, the $(K+1)$-th eigenvalue may be larger than the K-th eigenvalue, thus the $(K+1)$-th eigenvector falls into the first K principal components. To address this issue, one can choose to track a few more eigenvectors. And after each update, one reorders the eigenvalues and the eigenvectors.

To summarize, the overall method to partition the graph is given as follows. Consider a graph $\mathcal{G}(\mathcal{B},\mathcal{U},\mathcal{E})$, where $\mathcal{B}\cup\mathcal{U} = \mathcal{V}$ and $\mathcal{B}\cap\mathcal{U} = \varnothing$, to be partitioned, which has the adjacency matrix $\mathbf{W}$ and combinatorial Laplacian matrix $\mathbf{L}$. A

series of perturbations $\{\Delta\mathbf{W}_t\}_{t=1,2,...}$ are defined towards the graph brought by variations of the users' connection status. The partitions $\{\mathcal{C}_m\}_{m=1,...,K}$ defined by J_2 and the related indicator vectors $\{\mathbf{c}_m\}_{m=1,...,K}$ of the graph can then be updated by Algorithm 8 below when the graph is changing. Value I can be tuned as a trade off between accuracy and time consumption. The variable M in Algorithm 8 should be slightly chosen larger than the number of clusters K.

Algorithm 8: Fast Graph Clusters Update

Data: Graph $\mathcal{G}(\mathcal{B},\mathcal{U},\mathcal{E})$, the number of clusters K, M, a series of perturbations $\{\Delta\mathbf{W}_t\}_{t=1,2,...}$, ϵ for Tikhonov regularization

Result: $\{\mathbf{c}_m\}_{m=1,...,K}$ and K updated clusters for $t = 1, 2, ...$

Initialization: Compute $\mathbf{W}_\mathrm{r}$ and $\mathbf{L}$ at time instant $t = 0$ and solve the related GEVP, obtaining $\{\mathbf{c}_m\}_{m=1,...,M}$ and $\{\lambda_m\}_{m=1,...,M}$

for *t = 1,2,...* **do**

  Compute $\mathbf{W}_\mathrm{r}$ and $\mathbf{L}$ at time instant t.

  Compute $\Delta\mathbf{W}_\mathrm{r}$ and $\Delta\mathbf{L}$ using $\mathbf{W}_\mathrm{r}$ and $\mathbf{L}$ at time instants t and $t-1$

  for *m = 1,2,...,M* **do**

    set $\Delta\mathbf{c}_m = \mathbf{0}$

    for *i = 1,2,...,I* **do**

      Compute $\Delta\lambda_m$ by Proposition 10

      Determine $\Delta\mathbf{c}_m$ using the current value of $\Delta\lambda_m$ by solving (6.2.15)

    end

    $\lambda_m = \lambda_m + \Delta\lambda_m$

    $\mathbf{c}_m = \mathbf{c}_m + \Delta\mathbf{c}_m$

  end

  Sort $\{\lambda_m\}_{m=1,...,M}$ in descending order

  Sort $\{\mathbf{c}_m\}_{m=1,...,M}$ by the order of $\{\lambda_m\}_{m=1,...,M}$, choose the first K vectors and stack them into the matrix $\mathbf{U}'$

  Treat each row of $\mathbf{U}'$ as coordinates and apply K-means to obtain the clusters

  When a large degeneration in cost function value is observed or the number of iteration reaches a threshold, go to the first step and reinitialize the update by solving a GEVP as discussed in Section 4.2.5

end

6.2.1.1 Dimension-varying Update

For the case where the dimension of the graph is varying, i.e., new base stations or users, we can extend the matrices and vectors with zeros and then follow the update procedure as introduced before. For example, when the original equation for the general eigenvalues is

$$\mathbf{W}_\mathrm{r}\mathbf{c}_m = \lambda_m\mathbf{L}\mathbf{c}_m \tag{6.2.16}$$

we can define a new system

$$\mathbf{W}_\mathrm{r}^*\mathbf{c}_m^* = \lambda_m^*\mathbf{L}^*\mathbf{c}_m^*, \tag{6.2.17}$$

where the new matrix $\mathbf{W}_{\mathrm{r}}^{*}$ is larger than $\mathbf{W}_{\mathrm{r}}$ in size. The above system can be updated by updating

$$\begin{bmatrix} \mathbf{W}_{\mathrm{r}} & \mathbf{0} \\ \mathbf{0} & 0 \end{bmatrix} \begin{bmatrix} \mathbf{c}_m \\ 0 \end{bmatrix} = \lambda_m \begin{bmatrix} \mathbf{L} & \mathbf{0} \\ \mathbf{0} & 0 \end{bmatrix} \begin{bmatrix} \mathbf{c}_m \\ 0 \end{bmatrix} \tag{6.2.18}$$

where

$$\Delta \mathbf{W}_{\mathrm{r}} = \mathbf{W}_{\mathrm{r}}^{*} - \begin{bmatrix} \mathbf{W}_{\mathrm{r}} & \mathbf{0} \\ \mathbf{0} & 0 \end{bmatrix} \tag{6.2.19}$$

$$\Delta \mathbf{L} = \mathbf{L}^{*} - \begin{bmatrix} \mathbf{L} & \mathbf{0} \\ \mathbf{0} & 0 \end{bmatrix} \tag{6.2.20}$$

using the algorithm proposed before.

6.2.2 MinMaxCut: Derivation of Update

In the bipartite graph case, we have

$$\mathbf{B}\mathbf{u}_m = \mu_m \mathbf{D}_{\mathrm{b}} \mathbf{b}_m, \tag{6.2.21}$$

$$\mathbf{B}^{\top}\mathbf{b}_m = \mu_m \mathbf{D}_{\mathrm{u}} \mathbf{u}_m. \tag{6.2.22}$$

Considering the $\mathbf{B}$ matrix is perturbed by $\Delta\mathbf{B}$, which denotes the change brought by its time-varying property, we would like to find the generalized eigenvectors and eigenvalues of the perturbated eigensystem

$$(\mathbf{B} + \Delta\mathbf{B})(\mathbf{u}_m + \Delta\mathbf{u}_m) = (\mu_m + \Delta\mu_m)(\mathbf{D}_{\mathrm{b}} + \Delta\mathbf{D}_{\mathrm{b}})(\mathbf{b}_m + \Delta\mathbf{b}_m), \tag{6.2.23}$$

$$(\mathbf{B} + \Delta\mathbf{B})^{\top}(\mathbf{b}_m + \Delta\mathbf{b}_m) = (\mu_m + \Delta\mu_m)(\mathbf{D}_{\mathrm{u}} + \Delta\mathbf{D}_{\mathrm{u}})(\mathbf{u}_m + \Delta\mathbf{u}_m). \tag{6.2.24}$$

As we have discussed before, in the bipartite graph case we can partition the base stations and then assign the users, which only requires updating $\mathbf{b}_m$. In order to do so, we transform (6.2.22) such that we can get rid of $\mathbf{u}_m$.

We left multiply $\mathbf{B}^{\top}\mathbf{b}_m$ with $\mathbf{B}\mathbf{D}_{\mathrm{u}}^{-1}$, where $\mathbf{D}_{\mathrm{u}}$ is ensured to be invertible as all nodes have at least one neighbour, and we will obtain

$$\mathbf{B}\mathbf{D}_{\mathrm{u}}^{-1}\mathbf{B}^{\top}\mathbf{b}_m = \mathbf{B}\mathbf{D}_{\mathrm{u}}^{-1}\mu_m \mathbf{D}_{\mathrm{u}}\mathbf{u}_m \tag{6.2.25}$$

$$= \mu_m \mathbf{B}\mathbf{u}_m \tag{6.2.26}$$

$$= \mu_m^2 \mathbf{D}_{\mathrm{b}}\mathbf{b}_m, \tag{6.2.27}$$

which is a GEVP *w.r.t* the eigenpair (μ_m^2,$\mathbf{b}_m$), denoted as

$$\hat{\mathbf{B}}\mathbf{b}_m = \sigma_m \mathbf{D}_{\mathrm{b}}\mathbf{b}_m. \tag{6.2.28}$$

Its perturbed eigensystem is

$$(\hat{\mathbf{B}} + \hat{\Delta\mathbf{B}})(\mathbf{b}_m + \Delta\mathbf{b}_m) = (\sigma_m + \Delta\sigma_m)(\mathbf{D}_{\mathrm{b}} + \Delta\mathbf{D}_{\mathrm{b}})(\mathbf{b}_m + \Delta\mathbf{b}_m). \tag{6.2.29}$$

The update method for a GEVP has already been introduced in the adaptive solution for the second problem. Here, the update for (6.2.28) follows the same procedure. Therefore, we are not going to give all the details but just mention some important points.

Proposition 11. The $\Delta\sigma_m$ defined as the perturbation of the generalized eigenvalue of (6.2.28) can be determined by the following equation:

$$\Delta\sigma_m = \frac{\mathbf{b}_m^\top(\hat{\Delta\mathbf{B}} - \sigma_m\Delta\mathbf{D}_\mathrm{b})(\mathbf{b}_m + \Delta\mathbf{b}_m)}{\mathbf{b}_m^\top(\mathbf{D}_\mathrm{b} + \Delta\mathbf{D}_\mathrm{b})(\mathbf{b}_m + \Delta\mathbf{b}_m)} \tag{6.2.30}$$

Similarly, the following proposition can be implemented to determine the perturbation $\Delta\mathbf{b}_m$ of $\mathbf{b}_m$.

Proposition 12. Tikhonov regularization. The $\Delta\mathbf{b}_m$ brought by the time-varying property of the graph can be determined by solving the following optimization problem:

$$\min_{\Delta\mathbf{b}_m} ||\mathbf{K}\Delta\mathbf{b}_m - \mathbf{h}||_2^2 + \epsilon||\Delta\mathbf{b}_m||_2^2, \tag{6.2.31}$$

where $\mathbf{K} = (\hat{\mathbf{B}} + \hat{\Delta\mathbf{B}} - (\sigma_m + \Delta\sigma_m)(\mathbf{D}_\mathrm{b} + \Delta\mathbf{D}_\mathrm{b}))$ and $\mathbf{h} = (\Delta\sigma_m\mathbf{D}_\mathrm{b} + \sigma_m\Delta\mathbf{D}_\mathrm{b} + \Delta\sigma_m\Delta\mathbf{D}_\mathrm{b} - \hat{\Delta\mathbf{B}})\mathbf{b}_m$. The $\hat{\Delta\mathbf{B}}$ can be obtained by the subtraction between the $\hat{\mathbf{B}}$ at the new time instant and the previous $\hat{\mathbf{B}}$.

The closed-form solution towards (6.2.31) is given by:

$$(\mathbf{K}^\top\mathbf{K} + \epsilon\mathbf{I})\Delta\mathbf{b}_m = \mathbf{K}^\top\mathbf{h} \tag{6.2.32}$$

which can be solved efficiently by conjugate gradient descent as shown in Algorithm 10 of Appendix A.

This concludes the update procedure for the generalized eigenvectors. And similarly, when there are new base stations and the dimension of the matrix changes, we can follow the update method described in Section 6.2.1.1. Regarding the case where the number of users changes, since the dimension of $\hat{\mathbf{B}}$ will not change, we can follow the regular update way introduced in Proposition 12.

We can now summarize the whole procedure as follows. Consider we have a bipartite graph $\mathcal{G}_\mathrm{b}(\mathcal{B},\mathcal{U},\mathbf{B})$ to be partitioned and a series of perturbations $\{\Delta\mathbf{B}_t\}_{t=1,2,\ldots}$ towards the bipartite graph brought by either new users, base stations or variations of the users' connection status. $\{\mathbf{b}_m\}_t$ are the first K eigenvectors at each time instant which are defined by (6.2.28). Then the $\{\mathbf{b}_m\}_t$ can be updated by Algorithm 9 and correspondingly the clusters of the bipartite graph. Value I can be tuned as a trade-off between accuracy and time consumption. As we have discussed before, the variable M in Algorithm 9 should be chosen slightly larger than the number of clusters K.

6.3 Complexity Analysis

In this section, we will analyze and compare the time complexity of our proposed algorithms, especially for the devised update method. We consider that we have matrices $\hat{\mathbf{B}}$ and $\mathbf{D}_\mathrm{b}$, or $\mathbf{W}_\mathrm{r}$ and $\mathbf{L}$ for the new time instant at hand, meaning that we don't consider computing them but only compute the update steps.

First, let's consider the user assignment method. In this case, we avoid the spectral embedding-based users partitioning. In the regular spectral clustering,

Algorithm 9: Fast Bipartite Graph Clusters Update

Data: Bipartite Graph $\mathcal{G}_{\mathrm{b}}(\mathcal{B},\mathcal{U},\mathbf{B})$, the number of clusters K, M, a series of perturbations $\{\Delta\mathbf{B}_t\}_{t=1,2,\ldots}$, ϵ for Tikhonov regularization

Result: $\{\mathbf{b}_m\}_{m=1,\ldots,K}$ and K updated clusters for $t = 1, 2, ...$

1 **Initialization**: Compute $\{\mathbf{b}_m\}_{m=1,\ldots,M}$ and $\{\sigma_m\}_{m=1,\ldots,M}$ towards (6.2.28)
2 **for** *t = 1,2,...* **do**
3 Compute $\Delta\hat{\mathbf{B}}$ and $\Delta\mathbf{D}_{\mathrm{b}}$ using $\hat{\mathbf{B}}$ at the current and the previous time instants
4 **for** *m = 1,2,...,M* **do**
5 set $\Delta\mathbf{b}_m = \mathbf{0}$
6 **for** *i = 1,2,...,I* **do**
7 Compute $\Delta\sigma_m$ by Proposition 10
8 Determine $\Delta\mathbf{b}_m$ using the current value of $\Delta\sigma_m$ by solving (6.2.32)
9 **end**
10 $\sigma_m = \sigma_m + \Delta\sigma_m$
11 $\mathbf{b}_m = \mathbf{b}_m + \Delta\mathbf{b}_m$
12 **end**
13 Sort $\{\sigma_m\}_{m=1,\ldots,M}$ in the descending order
14 Sort $\{\mathbf{b}_m\}_{m=1,\ldots,M}$ by the order of $\{\sigma_m\}_{m=1,\ldots,M}$, choose the first K vectors and stack them into the matrix $\mathbf{U}'$
15 Treat each row of $\mathbf{U}'$ as coordinates and apply K-means to obtain the clusters for base stations
16 Apply user assignment method by solving (4.1.21) for each user
17 Combine the labels for users and base stations to obtain the clusters for the bipartite graph
18 When the accumulated error cannot be ignored or a large degeneration in cost function value is observed, go to the first step and reinitialize the update by solving (6.2.28)
19 **end**

one should compute the EVD of the Laplacian matrix of the bipartite graph and then perform the K-means. In the proposed method, in order to partition the bipartite graph, we first partition the graph composed of base stations and then assign users to each cluster. Partitioning base stations first allows us to work on a graph with a smaller size, giving a speedup. The complexity of the user assignment method is linear in the number of users $|\mathcal{U}|$, which is preferable as well.

Then let's consider the proposed update method and take Algorithm 8 as an example. The update steps start from line 5 to line 13, which can be roughly divided into two parts, that is, computing $\Delta\lambda_m$ and computing $\Delta\mathbf{c}_m$ by conjugate gradient descent. Computing $\Delta\lambda_m$ uses Proposition 10. We would like to mention that in our case both $\mathbf{W}_{\mathrm{r}}$ and $\mathbf{L}$ are sparse. We denote the number of non-zero elements of matrices $\mathbf{W}_{\mathrm{r}}$ and $\mathbf{L}$ as $\mathcal{O}(|\mathcal{E}|)$ and $\mathcal{O}(|\mathcal{V}| + |\mathcal{E}|)$, respectively.

For better illustration of determining $\Delta\lambda$, we mention the update equation here again:

$$\Delta\lambda = \frac{\mathbf{c}^\top(\Delta\mathbf{W}_r - \lambda\Delta\mathbf{L})(\mathbf{c} + \Delta\mathbf{c})}{\mathbf{c}^\top(\mathbf{L} + \Delta\mathbf{L})(\mathbf{c} + \Delta\mathbf{c})}. \tag{6.3.1}$$

The number of computations required for the numerator is upper bounded by $\mathcal{O}(|\mathcal{V}| + |\mathcal{E}|)$. First, computing $(\Delta\mathbf{W}_\mathrm{r} - \lambda\Delta\mathbf{L})$ requires at most $\mathcal{O}(|\mathcal{V}| + |\mathcal{E}|)$ operations. This is because the matrix $\Delta\mathbf{L}$ is usually sparser than $\mathbf{L}$ since $\Delta\mathbf{L}$ only preserves the terms related to the changing edges and their terminal nodes. If the dynamic property of the graph is brought by a movement of all the users, the number of non-zero elements is upper bounded by $\mathcal{O}(|\mathcal{V}| + |\mathcal{E}|)$ where every user moves. Then the multiplication between the dense vector $\mathbf{c}$ and the sparse matrix $(\Delta\mathbf{W}_\mathrm{r} - \lambda\Delta\mathbf{L})$ requires at most $\mathcal{O}(|\mathcal{V}| + |\mathcal{E}|)$ operations. After that, computing the inner product between $\mathbf{c}^\top(\Delta\mathbf{W}_\mathrm{r} - \lambda\Delta\mathbf{L})$ and $(\mathbf{c} + \Delta\mathbf{c})$ requires $\mathcal{O}(|\mathcal{V}|)$ operations where the length of the $\mathbf{c}$ or $\Delta\mathbf{c}$ is $|\mathcal{V}|$. To sum up, computing the numerator requires $\mathcal{O}(|\mathcal{V}| + |\mathcal{E}|)$ operations. Similarly, computing the denominator requires $\mathcal{O}(|\mathcal{V}| + |\mathcal{E}|)$ operations as well.

What about the complexity of determining $\Delta\mathbf{c}$? We first have to compute matrix $\tilde{\mathbf{K}}$. It can be seen that this step requires $\mathcal{O}(|\mathcal{V}| + |\mathcal{E}|)$ operations. Computing vector $\mathbf{h}$ requires $\mathcal{O}(|\mathcal{V}| + |\mathcal{E}|)$ as well since matrix $(\Delta\lambda\mathbf{L} + \lambda\Delta\mathbf{L} + \Delta\lambda\Delta\mathbf{L} - \Delta\mathbf{W}_\mathrm{r})$ has $\mathcal{O}(|\mathcal{V}| + |\mathcal{E}|)$ non-zero elements. After that, we have to solve the *Tikhonov regularization* problem as described in (6.2.15). Computing $\tilde{\mathbf{K}}^\top\tilde{\mathbf{K}}$ requires $\mathcal{O}(|\mathcal{V}| \times (|\mathcal{V}| + |\mathcal{E}|))$ operations. As we can see, we introduce a quadratic complexity here. However, in the conjugate gradient, only the matrix-vector multiplication is required. Therefore, instead of computing $\tilde{\mathbf{K}}^\top\tilde{\mathbf{K}}$ first, we can change the order of multiplication by performing the product of $\tilde{\mathbf{K}}$ and the vector first and then multiplied with $\tilde{\mathbf{K}}^\top$. The product of a sparse $\tilde{\mathbf{K}}$ and a dense $\mathbf{h}$ asks for $\mathcal{O}(|\mathcal{V}| + |\mathcal{E}|)$ operations. Finally, the complexity of the conjugate gradient descent is $\mathcal{O}(T \times (|\mathcal{V}| + |\mathcal{E}|))$ where T is the number of iteration [35]. To sum up, the complexity of this step is $\mathcal{O}(|\mathcal{V}| + |\mathcal{E}|)$.

To perform one update, $\mathcal{O}(|\mathcal{V}| + |\mathcal{E}|)$ operations are required. If we take the number of components M and the number of iterations I into consideration, to update the generalized eigenvectors, $\mathcal{O}(M \times I \times (|\mathcal{V}| + |\mathcal{E}|))$ operations are needed. When these two numbers are small (which is the case in practical implementation), we can say that the complexity is linear in the number of non-zero entries of the Laplacian, which is much smaller compared with the cubic complexity of the naive exact EVD implementation. In the simulation results, we can observe that time is saved if we only update several components instead of computing the EVD every time from scratch.

Regarding the update procedure for the first problem in Algorithm 9, since we partition base stations first, the matrices used for spectral clustering have smaller size. Although the bipartite graph itself is sparse, these matrices, e.g., $\mathbf{K}$, usually are not sparse enough. Sparse methods can be slower when applied to a non-sparse matrix. Therefore, computing $\mathbf{K}$ and vector multiplication in the conjugate gradient descent method should be based on the full matrix, having quadratic complexity $\mathcal{O}(|\mathcal{B}|^2)$. And this update method will bring less gain. However, when the size of the network becomes larger, one can expect more gain.

Bibliography

[1] B. Yang, "Projection approximation subspace tracking," *IEEE Transactions on Signal processing*, vol. 43, no. 1, pp. 95–107, 1995.

[2] Y. Hua, Y. Xiang, T. Chen, K. Abed-Meraim, and Y. Miao, "A new look at the power method for fast subspace tracking," *Digital Signal Processing*, vol. 9, no. 4, pp. 297–314, 1999.

[3] K. Abed-Meraim, A. Chkeif, and Y. Hua, "Fast orthonormal past algorithm," *IEEE Signal processing letters*, vol. 7, no. 3, pp. 60–62, 2000.

[4] N. Lassami, K. Abed-Meraim, and A. Aïssa-El-Bey, "Low cost subspace tracking algorithms for sparse systems," in *2017 25th European Signal Processing Conference (EUSIPCO)*. IEEE, 2017, pp. 1400–1404.

[5] S. Attallah and K. Abed-Meraim, "A fast adaptive algorithm for the generalized symmetric eigenvalue problem," *IEEE Signal Processing Letters*, vol. 15, pp. 797–800, 2008.

[6] J. Yang, X. Chen, and H. Xi, "Fast adaptive extraction algorithm for multiple principal generalized eigenvectors," *International journal of intelligent systems*, vol. 28, no. 3, pp. 289–306, 2013.

[7] M. Baumann, U. Helmke, and J. H. Manton, "Reliable tracking algorithms for principal and minor eigenvector computations," in *Proceedings of the 44th IEEE Conference on Decision and Control*. IEEE, 2005, pp. 7258–7263.

[8] Z. Wen, C. Yang, X. Liu, and Y. Zhang, "Trace-penalty minimization for large-scale eigenspace computation," *Journal of Scientific Computing*, vol. 66, no. 3, pp. 1175–1203, 2016.

[9] Y. Miao and Y. Hua, "Fast subspace tracking and neural network learning by a novel information criterion," *IEEE Transactions on Signal Processing*, vol. 46, no. 7, pp. 1967–1979, 1998.

[10] M. Gu and S. C. Eisenstat, "A stable and fast algorithm for updating the singular value decomposition," 1993.

[11] M. Brand, "Fast low-rank modifications of the thin singular value decomposition," *Linear algebra and its applications*, vol. 415, no. 1, pp. 20–30, 2006.

[12] ——, "Fast online svd revisions for lightweight recommender systems," in *Proceedings of the 2003 SIAM international conference on data mining*. SIAM, 2003, pp. 37–46.

[13] S. Bartelmaos and K. Abed-Meraim, "Fast adaptive algorithms for minor component analysis using householder transformation," *Digital Signal Processing*, vol. 21, no. 6, pp. 667–678, 2011.

[14] H. Sakai and K. Shimizu, "A new adaptive algorithm for minor component analysis," *Signal Processing*, vol. 71, no. 3, pp. 301–308, 1998.

[15] M. Thameri, K. Abed-Meraim, and A. Belouchrani, "Low complexity adaptive algorithms for principal and minor component analysis," *Digital Signal Processing*, vol. 23, no. 1, pp. 19–29, 2013.

[16] L. Yang, "Design and analysis of adaptive noise subspace estimation algorithms," 2009.

[17] S. Chan, Z. Zhang, and Y. Zhou, "A new adaptive kalman filter-based subspace tracking algorithm and its application to doa estimation," in *2006 IEEE International Symposium on Circuits and Systems*. IEEE, 2006, pp. 4–pp.

[18] J. H. Wilkinson, *The algebraic eigenvalue problem*. Clarendon press Oxford, 1965, vol. 87.

[19] B. Champagne, "Adaptive eigendecomposition of data covariance matrices based on first-order perturbations," *IEEE Transactions on Signal Processing*, vol. 42, no. 10, pp. 2758–2770, 1994.

[20] R. S. Varga, *Geršgorin and his circles*. Springer Science & Business Media, 2010, vol. 36.

[21] C. Chen and H. Tong, "Fast eigen-functions tracking on dynamic graphs," in *Proceedings of the 2015 SIAM international conference on data mining*. SIAM, 2015, pp. 559–567.

[22] P. D. Cha and A. Shin, "Perturbation methods for the eigencharacteristics of symmetric and asymmetric systems," *Shock and Vibration*, vol. 2018, 2018.

[23] H. Ning, W. Xu, Y. Chi, Y. Gong, and T. S. Huang, "Incremental spectral clustering by efficiently updating the eigen-system," *Pattern Recognition*, vol. 43, no. 1, pp. 113–127, 2010.

[24] G. W. Stewart, "Matrix perturbation theory," 1990.

[25] C. S. MacInnes and R. J. Vaccaro, "Tracking directions-of-arrival with invariant subspace updating," *Signal processing*, vol. 50, no. 1-2, pp. 137–150, 1996.

[26] A. Bouchachia and M. Prossegger, "Incremental spectral clustering," in *Learning in Non-Stationary Environments*. Springer, 2012, pp. 77–99.

[27] C. Valgren, T. Duckett, and A. Lilienthal, "Incremental spectral clustering and its application to topological mapping," in *Proceedings 2007 IEEE International Conference on Robotics and Automation*. IEEE, 2007, pp. 4283–4288.

[28] C. Valgren and A. Lilienthal, "Incremental spectral clustering and seasons: Appearance-based localization in outdoor environments," in *2008 IEEE International Conference on Robotics and Automation*. IEEE, 2008, pp. 1856–1861.

[29] T. Kong, Y. Tian, and H. Shen, "A fast incremental spectral clustering for large data sets," in *2011 12th International Conference on Parallel and Distributed Computing, Applications and Technologies*. IEEE, 2011, pp. 1–5.

[30] A. Ng, M. Jordan, and Y. Weiss, "On spectral clustering: Analysis and an algorithm," *Advances in neural information processing systems*, vol. 14, pp. 849–856, 2001.

[31] C. Gupta and R. Grossman, "Genic: A single pass generalized incremental algorithm for clustering," in *Proceedings of the 2004 SIAM International Conference on Data Mining*. SIAM, 2004, pp. 147–153.

[32] S. Guha and N. Mishra, "Clustering data streams," in *Data stream management*. Springer, 2016, pp. 169–187.

[33] M. Charikar, C. Chekuri, T. Feder, and R. Motwani, "Incremental clustering and dynamic information retrieval," *SIAM Journal on Computing*, vol. 33, no. 6, pp. 1417–1440, 2004.

[34] Y. Jia, "Online spectral clustering on network streams," Ph.D. dissertation, University of Kansas, 2012.

[35] J. R. Shewchuk *et al.*, "An introduction to the conjugate gradient method without the agonizing pain," 1994.

7 Numerical Results

7.1 Problem 1: Numerical Simulations

In this section, we will first introduce the way to generate the synthetic data. And then the values chosen in the update algorithms will be provided. Finally, the simulation results containing tracking performance and partitioning results will be provided.

7.1.1 Simulation Details

The bipartite graph and its $\mathbf{B}$ matrix are generated from the data of a simulated cellular network. This cellular network has 37 base stations and 3 hotspots with a Gaussian shape, located at cell 9, 13, and 17, as shown in Fig. 7.1. The shape of each hotspot is determined by a Gaussian probability distribution. Each hotspot has 400 users. In total, the 1200 users yield the initial graph. A red circle is a base station and a blue dot is a user. The inter-site distance (ISD) which is the distance between two base stations is 1 *kilometer*. Here we assume that there are enough resources to maintain the connection to every user such that there will be no issues in resources allocation/Physical Resource Blocks (PRBs) distribution. In Fig. 7.1, we give the initial clusters as well. The initial clusters are generated by partitioning base stations first and then applying user assignment method. Partitioning base stations is based on the generalized eigenvectors of $\hat{\mathbf{B}}$ computed by EVD or the vector transformed by the left singular vector of $\tilde{\mathbf{B}}$ left-multiplied with $\mathbf{D}_{\mathrm{b}}^{-\frac{1}{2}}$.

The $\mathbf{B}$ matrix is constructed as follows. First, a path-loss/gain model is used to compute the received power of each user:

$$PL(dB) = 100 + 30 \log (distance) \tag{7.1.1}$$

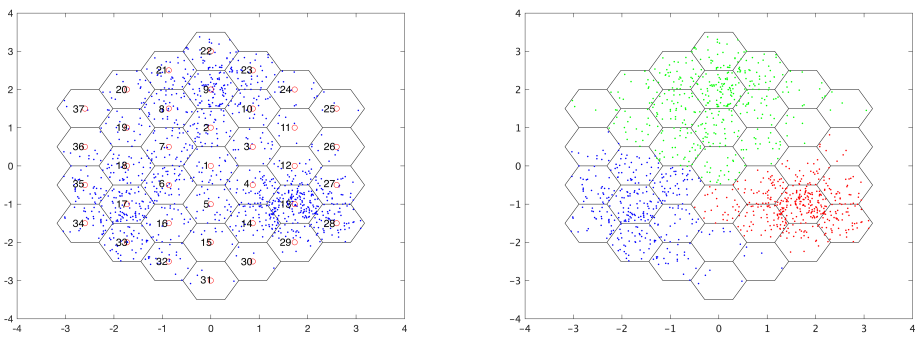


Figure 7.1: Cellular Network and Initial Clusters

where the distance is in *kilometer*.

The transmitting power is 30 dBm. Here we make the assumption that all the antennas are omnidirectional and each user receives interference from the tier 1 sites, e.g., users in the center cell will be interfered by power from the surrounding 6 cells. After obtaining the down-link received power, we can derive the *co-channel signal-to-interference-ratio (SIR)* of each down-link connection for each user. Finally, as defined in the LTE system specification [1], the spectral efficiency of each link can be determined, which serves as the weight of each link, that is, the value of the corresponding entry in $\mathbf{B}$.

In order to show the performance of the considered updating methods, we define the following metrics including *Direction-Cosine*, *Subspace Distance*, and *Absolute Error* between estimated and the true eigenvectors and eigenvalues. Before defining such metrics, we first give the notations of the estimated and true values.

The estimated subspace is $\mathbf{U}' = [\mathbf{b}'_1, \mathbf{b}'_2, ..., \mathbf{b}'_M]$ containing the estimated eigenvectors on its columns. The true subspace is $\mathbf{U} = [\mathbf{b}_1, \mathbf{b}_2, ..., \mathbf{b}_M]$ containing the true eigenvectors on its columns. The estimated eigenvalues are $\{\sigma'_m\}_{m=1,...,M}$ while the true ones are $\{\sigma_m\}_{m=1,...,M}$. As we have introduced, we track M eigenvectors and eigenvalues but only use the first K to compute errors and perform clustering. $||\cdot||_F$ means the Frobenius norm. We would like to clarify that in the figures we don't exactly show the *Direction-Cosine* but $(1 - $ *Direction-Cosine*$)$ such that all the three metrics start from 0 which gives better illustration.

- Direction-Cosine (DC): $\xi(\mathbf{U}', \mathbf{U}) = \frac{1}{K}\sum_{m=1}^{K} \frac{|\mathbf{b}_m^\top \mathbf{b}'_m|}{||\mathbf{b}_m||||\mathbf{b}'_m||}$
- Subspace Distance (SD): $\xi(\mathbf{U}'_{1:K}, \mathbf{U}_{1:K}) = ||\frac{\mathbf{U}'_{1:K}\mathbf{U}'^\top_{1:K}}{tr(\mathbf{U}'^\top_{1:K}\mathbf{U}'_{1:K})} - \frac{\mathbf{U}_{1:K}\mathbf{U}^\top_{1:K}}{tr(\mathbf{U}^\top_{1:K}\mathbf{U}_{1:K})}||_F$
- Absolute Error (AE): $\xi(\{\sigma_m\}, \{\sigma'_m\}) = \sum_{m=1}^{K} |\sigma_m - \sigma'_m|$

In order to show the error-reduction performance of the proposed updating method, we compare the *DC*, *SD* and *AE w.r.t with update* case and *without update* case. For both cases, the ways to implement Algorithm 9 are almost the same. First, we have a initial graph with $\mathbf{B}$ matrix at $t = 0$ and we can correspondingly obtain $\tilde{\mathbf{B}}$ and $\hat{\mathbf{B}}$. Then we compute their exact SVD or EVD to obtain the initial singular vectors or eigenvectors and then form $\mathbf{U}_{t=0}$, and correspondingly the initial clusters by K-means. At $t = 1$, some variations $\Delta\mathbf{B}_{t=1}$ are generated, which can be users' movement or the appearances of new users. We can obtain the $\mathbf{U}'_{t=1}$ as described in Algorithm 9, which is related to *with update* case, or do nothing and obtain $\mathbf{U}'_{t=1} = \mathbf{U}_{t=0}$, which is *without update* case. And the true $\mathbf{U}_{t=1}$ is also computed. After obtaining *with update* $\mathbf{U}'_{t=1}$, *without update* $\mathbf{U}'_{t=1}$ and $\mathbf{U}_{t=1}$, the difference between *with update* $\mathbf{U}'_{t=1}$ and $\mathbf{U}_{t=1}$, *without update* $\mathbf{U}'_{t=1}$ and $\mathbf{U}_{t=1}$ can be derived. Since we track the generalized eigenvectors, we can expect the *with update* difference is smaller than *without update* difference. Finally, the aforementioned steps are implemented for $t = 2, 3, ...$ And the same logic is used for the eigenvalues.

Furthermore, since our goal is to adaptively update the clusters, we should perform clustering algorithm using the estimated and the true vectors at each

time instant. There are several options. First, we can rely on Algorithm 4 and do spectral clustering, called `BSC`. Second, we can rely on the Exact generalized eigenvectors $\mathbf{U}$ of $\hat{\mathbf{B}}$ and then follow the User Assignment method. Partitioning base stations is based on K-means. This option is called `EUAK`. Third, we can rely on the *with Update* $\mathbf{U}'$ and the User Assignment method, called `UUAK`. The last option is we can rely on *Without update* $\mathbf{U}'$ and the User Assignment method, called `WUAK`. The comparison between the first and the second option can demonstrate the benefit of using user assignment method. The comparison between the second and the third option tells us how well this updating Algorithm 9 performs. And the comparison between the third and the last illustrates how much error we can reduce by Algorithm 9. And you can see these comparisons in the simulation results.

In the simulations, ϵ is 10^{-3} and I is 2. The i_{max} is set to be the length of $\tilde{\mathbf{h}}$. As described before, we have 3 hotspots. Then we have to track the first 3 eigenvectors corresponding to the largest eigenvalues. However, the eigenvector corresponding to the largest eigenvalue is always an all-one vector, the tracking is unnecessary. Therefore, in the implementation, we don't track this eigenvector to reduce the computation. Furthermore, we set M to 5, that is, we track two more eigenvectors to address the minor-to-principal issue of the eigenvector.

7.1.2 Simulation of users' movement

In this section, we simulate two scenarios. In the first scenario, all of 1200 users change their positions, which is achieved by randomly moving the user to another position within a square centered at the origin position. The side length of the square is 200 *meters*. This would change the bipartite graph. Since the status variation of a single user won't change the column subspace of $\mathbf{B}$ greatly, when the graph is perturbed by the movements of 20 users, we update the eigenvectors and eigenvalues. And this forms 60 time instants and a series of $\{\Delta\mathbf{B}_t\}_{t=1,2,...,60}$. At $t = m$ for $m = 1, 2, ..., 60$, according to $\{\Delta\mathbf{B}_t\}_{t=m}$, we update the generalized eigenvectors of $\hat{\mathbf{B}}$ and perform clustering. In a word, we gradually update the $\mathbf{U}'$ using a series of perturbations and finally obtain the $\mathbf{U}'$ for the graph after users' movement.

The cellular network after users' movement and the final clusters at $t = 60$ are shown in Fig. 7.2. And the tracking results are shown in Fig. 7.3 and 7.4. In Fig. 7.3, we provide the DC, SD and AE at each time instance. All the red curves are the results involving update procedure while the blue curves are not with the update procedure. In Fig. 7.4, we give the cost function values changing over time and the time consumption to obtain the clusters at each time instance.

The second scenario is that all the three hotspots rotate taking the center cell as the center. In the simulation, the perturbation brought by rotation is divided into 60 small perturbations, and updating is performed for each perturbation, following the same logic as before. The cellular network after hotspots' rotation and the final clusters at $t = 60$ are shown in Fig. 7.5. The tracking results are shown in Fig. 7.6 and 7.7. We illustrate the cost function values and time consumption here as well.

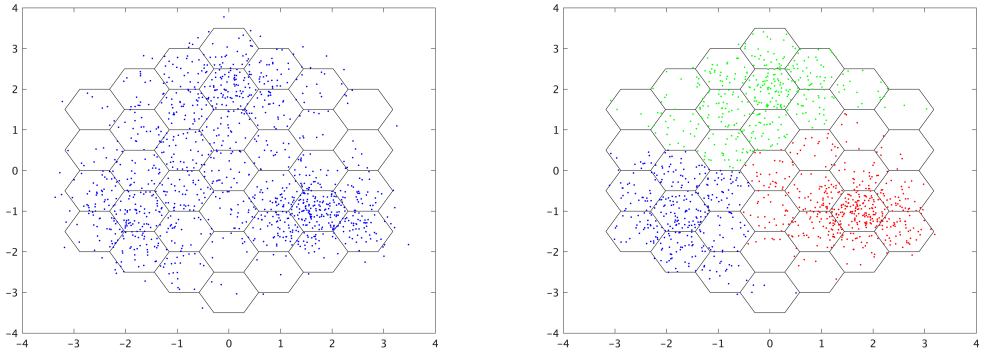


Figure 7.2: Left: Cellular network after randomly moving. Right: Clusters generated by updating the perturbed original network adaptively and user assignment method.

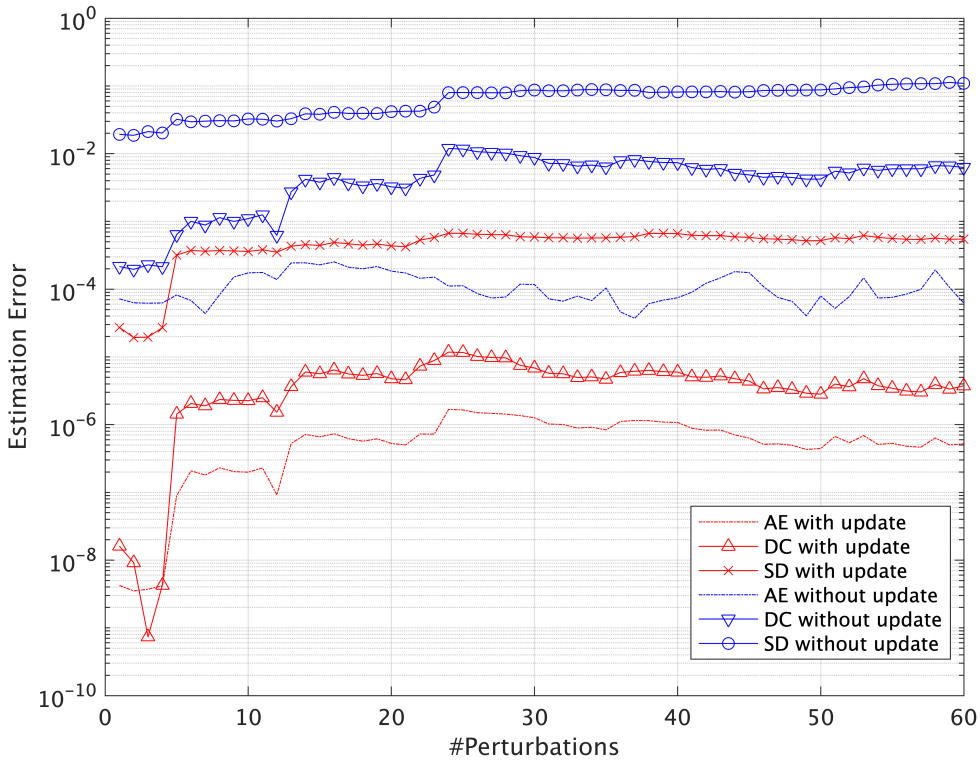


Figure 7.3: Tracking error when the graph is perturbed by users' movements.

7.1.3 Simulation of new users

In this section, we simulate the addition of a new hotspot. This hotspot contains 400 users and is located at cell 25 with a Gaussian shape as well. In the simulation, the perturbation brought by new users is divided into 80 small perturbations, and each perturbation is caused by the appearance of 5 new users. The cellular network after adding a hotspot and the final clusters at $t = 80$ are shown in Fig. 7.8. The tracking results are shown in Fig. 7.9 and 7.10. As we can see, in Fig. 7.9, there

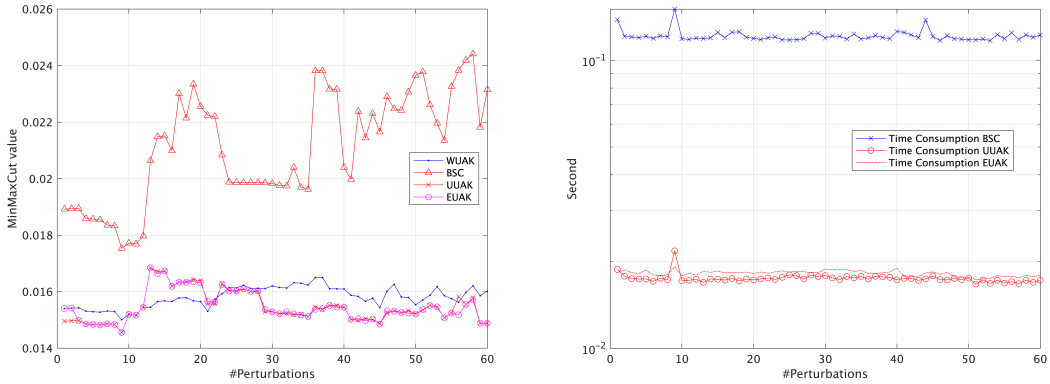


Figure 7.4: Left: **MinMaxCut** value varying with the graph perturbations by users' movements. Right: Time consumption of deriving the clusters after each perturbation.

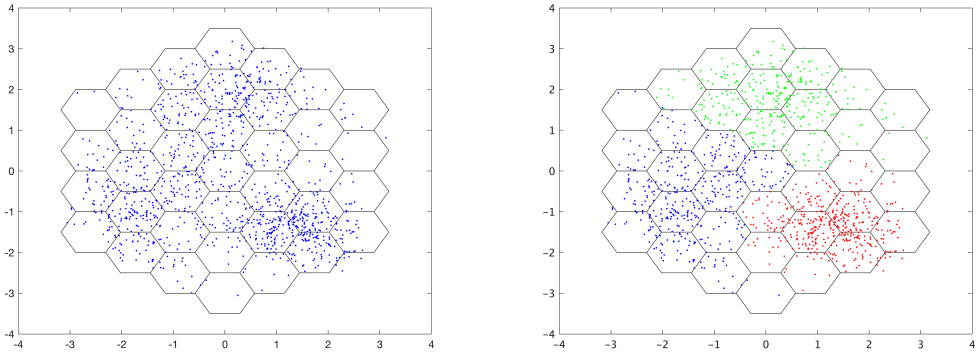


Figure 7.5: Left: Cellular network after rotating. Right: Clusters generated by updating the perturbed original network adaptively and user assignment method.

is a time when we cannot reduce error. This will be discussed in the next section.

7.1.4 Discussions

In this section, we will discuss the efficacy of the updating Algorithm 9 and the user assignment method.

Generally speaking, the updating algorithm can track the generalized eigenvectors and eigenvalues with less error compared with *without update* case, and with less time compared with computing EVD or SVD from scratch. As shown in Fig. 7.3 and 7.6, all the *with update* (red) curves are much lower than the corresponding *without update* (blue) ones, showing that the estimation error is reduced a lot and the tracking of eigenvectors and eigenvalues is accurate. Furthermore, as shown in the right part of Fig. 7.4, 7.7 and 7.10, compared with exact computation using `BSC`, the updating method is capable of saving time when the new subspace is required. But the update method saves little time when compared with `EUAK`. As we have already discussed in Section 6.3, in this case we implement full matrix instead of sparse matrix, therefore, we have quadratic complexity here. Although the complexity is lower than the EVD, since the matrix has small size ($|\mathcal{B}| = 37$),

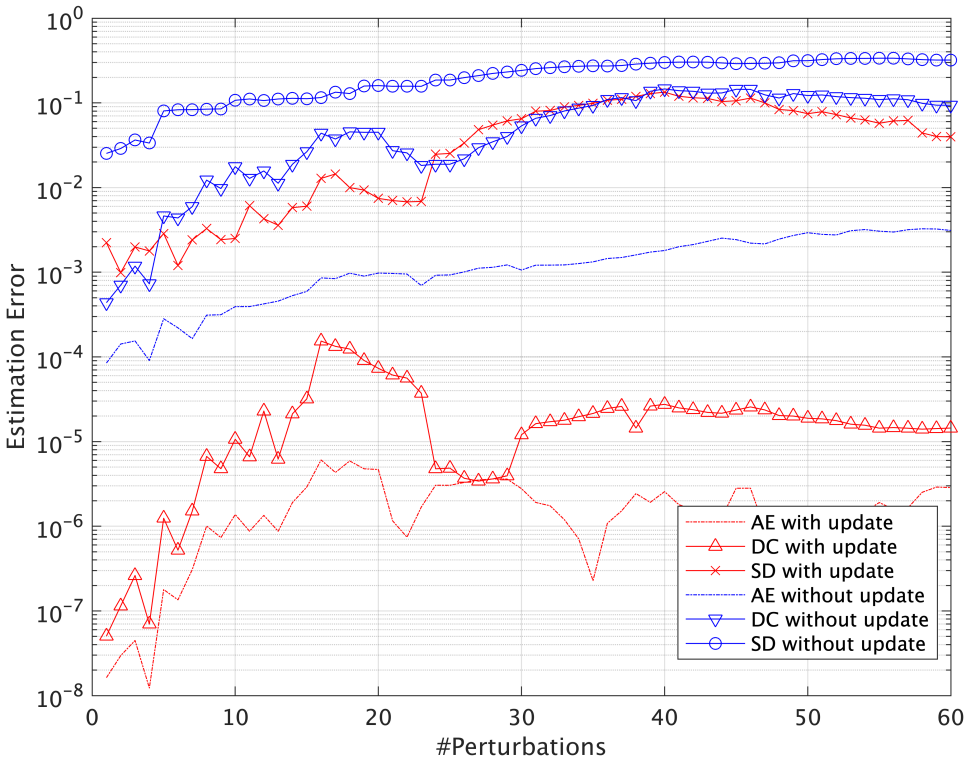


Figure 7.6: Tracking error when the graph is perturbed by hotspots' rotation.

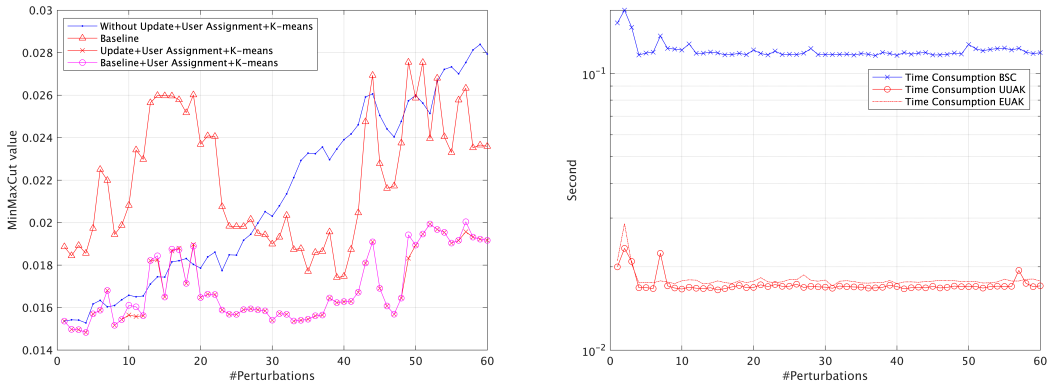


Figure 7.7: Left: **MinMaxCut** value varying with graph perturbations by hotspots' rotation. Right: Time consumption of deriving the clusters after each perturbation.

the amount of saved time is not obvious. But if the number of base stations becomes larger, due to the fact that the connection is local, we can expect a sparser $\hat{\mathbf{B}}$, where more time consumption can be reduced.

We can notice a special case in Fig. 7.9. The estimation error is low at the beginning. As the graph undergoes more perturbations, the estimation error becomes larger and at around 33rd perturbations, we cannot gain anything from the updating algorithm. But after this time instant, the error starts to reduce

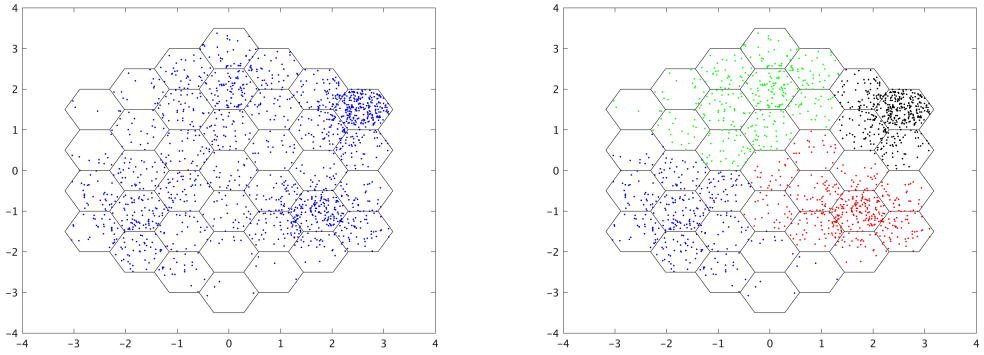


Figure 7.8: Left: Cellular network with a new hotspot. Right: Clusters generated by updating the perturbed original network adaptively and user assignment method.

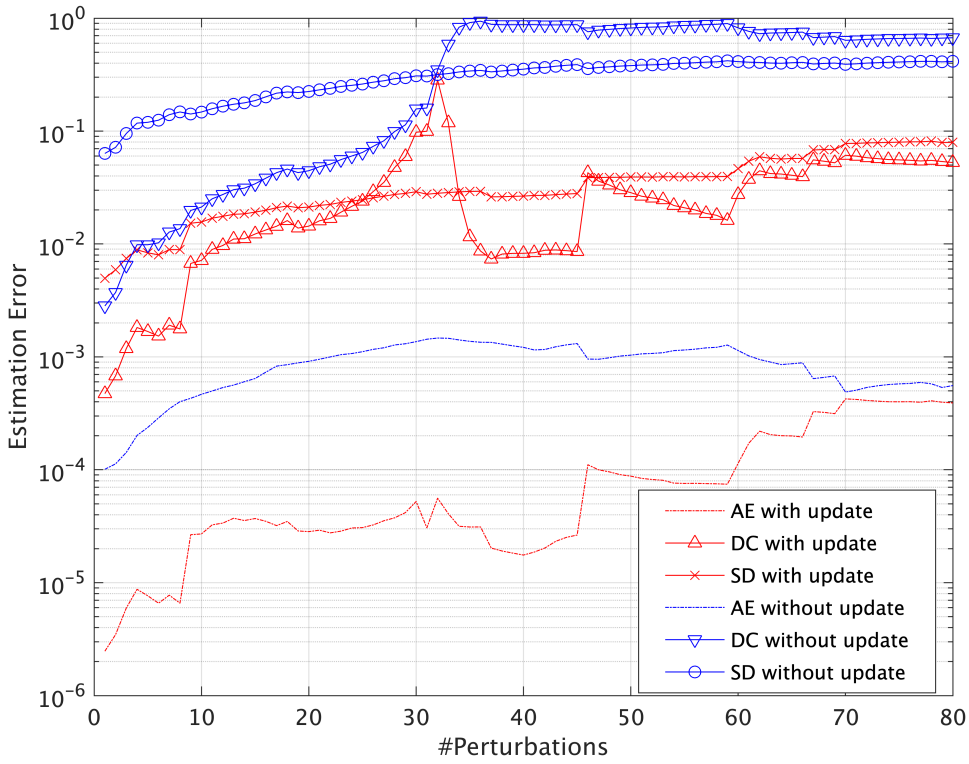


Figure 7.9: Tracking error when the graph is perturbed by a new hotspot.

rapidly and stays at a low level. This is because when the bipartite graph is evolving, the order of the eigenvalues changes. For example, the 3rd eigenvalue become larger than the 2nd one. But since we introduce a re-order process in Algorithm 9, we can correct this error. This is why the error reduces rapidly. And how the eigenvalues evolve is plotted in the Fig. 7.11, where we can see the order of eigenvalues changes and without the re-order process, we will lose track. Furthermore, although we assume that the number of clusters is known,

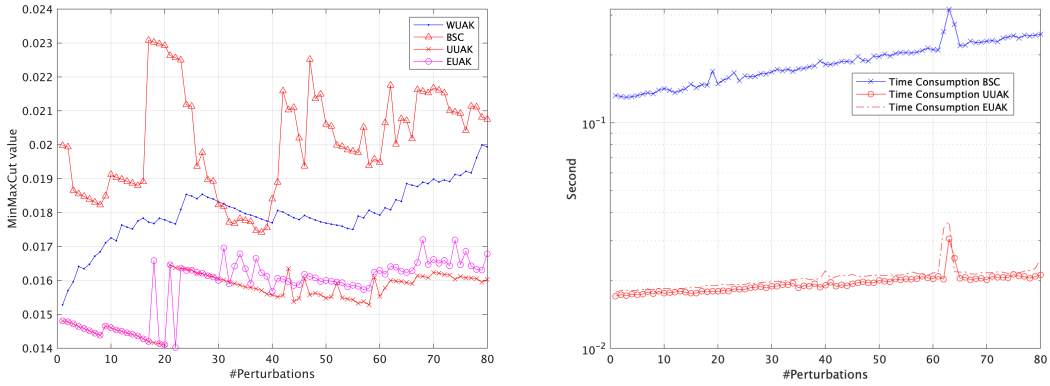


Figure 7.10: Left: **MinMaxCut** value varying with graph perturbations by a new hotspot. Right: Time consumption of deriving the clusters after each perturbation.

the appearance of a new cluster in the real world can be identified as well due to the high accuracy of the estimated eigenvalues, as shown in Fig. 7.11. In this figure, we show the leading 5 eigenvalues except for the largest one. A large gap appearing between the 4th and 5th can be observed, indicating the appearance of a new cluster.

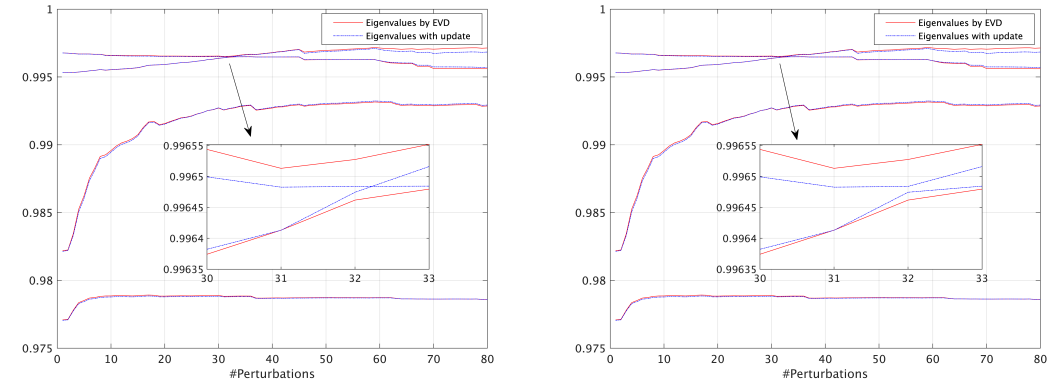


Figure 7.11: The distribution of eigenvalues in the new-user case. Left: No re-order process. Right: Re-order process

Now let's focus on the left parts of Fig. 7.4, 7.7 and 7.10. The comparison between `BSC` and `EUAK` illustrates that the proposed user assignment method can achieve lower cost function values than the Algorithm 4 does, which proves this method performs better than the Algorithm 4 does *w.r.t* minimizing the *MinMaxCut*. The comparison between `EUAK` and `UUAK` demonstrates that by using the estimated eigenvectors to partition the graph, we can obtain almost the same performance as using the exact eigenvactors. By comparing `UUAK` with `WUAK`, it is clear that it is necessary to perform the updating algorithm otherwise a degradation will appear *w.r.t* the *MinMaxCut* value.

To conclude, in the numerical simulations, we test the performance of our proposed *user assignment method* and update method. The *user assignment method* can achieve lower cost function value and less computation time, while in this case,

the update method is less preferable. But it can be expected that when the graph becomes larger and the sparsity can be fully exploited, more time consumption can be reduced.

7.2 Problem 2: Numerical Simulations

7.2.1 Simulation Details

The graph $\mathcal{G}(\mathcal{B},\mathcal{U},\mathcal{E})$ is generated from the data of a simulated cellular network as well. The cellular network has the same layout and parameters as the ones in the first problem simulation except for the location of each base station. In this case, since we have to define the edges between the same type of nodes, i.e., between a pair of base stations, we prefer heterogeneity in the distribution of the base stations. This heterogeneity is generated by adding uniformly distributed random variables to the coordinates of base stations. The heterogeneity in the layout of the base stations is achieved by randomly moving the node to another position within a square centered at the origin position. The side length of the square for the base station is 400 *meters*. The ISD before moving is 1 *kilometer* as well. The layout of the cellular network is shown in Fig. 7.12.

As we have motivated before, we can find such a model in the 4G or 5G cellular network. However, generating synthetic data is complicated. One has to make assumptions on the activities of users or consider the operations of handling users. Therefore, for the sake of demonstrating the proposed method, we take the following simple definition:

$$w_{i,j} = \frac{1}{\textit{Euclidean distance between } i \textit{ and } j} \tag{7.2.1}$$

which means that the weight of the edge will reduce and the connection will become weaker as the distance becomes larger.

Furthermore, the fact that the connection is local has to be considered. Here, we focus on the ϵ-ball neighborhood method. With a predefined threshold ϵ, this method only preserves edges having weights larger than ϵ, and the edges having weights smaller than ϵ are considered to be weak and eliminated. Regarding our case, this implies that if the distance is larger than the threshold, we will think of this link as a disconnection. However, since we have two types of edges, we should set two thresholds.

- For the edge between different base stations:

$$\epsilon = \frac{1}{1.1 \times \textit{ISD}} \tag{7.2.2}$$

  By taking this threshold, each base station only holds the connection to its neighboring base stations. The distance is multiplied by 1.1 due to the heterogeneity, guaranteeing the connection between each pair of neighboring base stations and in the meantime eliminating the connection between non-neighboring base stations.

- For the edge between user and base station:

$$\epsilon = \frac{1}{1.1 \times \textit{the radius of the cell}} \tag{7.2.3}$$

  By defining so, we can ensure that the cell center user only has a connection to the serving base station and the cell edge user only experiences interference from the neighboring cells. The distance is multiplied by 1.1 due to the heterogeneity as well.

Following the definitions, the adjacency matrix and the Laplacian matrix can be certainly produced. The initial layout of the test example graph and its clusters are illustrated in Fig. 7.12. The clusters are generated by the multiway partition method introduced in Algorithm 6.

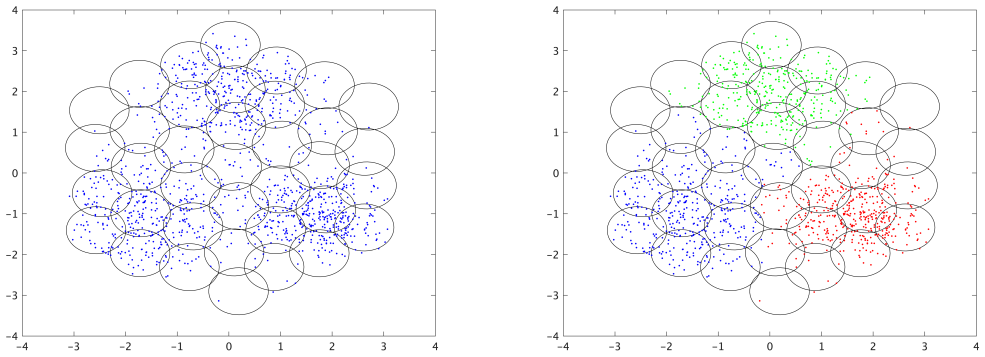


Figure 7.12: Left: Initial cellular network. Right: Generated clusters.

In order to show the error-reduction performance of the proposed update method, here we follow the same metrics as we have defined before. The way to implement Algorithm 8 is described. First, at $t = 0$, we have the initial graph layout with adjacency matrix $\mathbf{W}$ and the Laplacian matrix $\mathbf{L}$. We can correspondingly compute the matrices $\mathbf{W}_{\mathrm{r}}$ and $\mathbf{L}$ and solve the GEVP to obtain the initial $\mathbf{U}_{t=0}$. Then we can obtain the clusters by applying K-means. At $t = 1$, some perturbations $\Delta\mathbf{W}_{t=1}$ are brought to the graph and correspondingly the matrices $\mathbf{W}_{\mathrm{r}}$ and $\mathbf{L}$, denoted by $\Delta\mathbf{W}_{\mathrm{r}}$ and $\Delta\mathbf{L}$. We can turn to the Algorithm 8 to obtain the updated approximated $\mathbf{U}'_{t=1}$, which is called *with update* case and denoted as `UMP` (Updated Multi-way Partition) while not turning to the Algorithm 8 is called as *without update* case and denoted as `WMP` (Without update Multi-way Partition). The true $\mathbf{U}_{t=1}$ is also computed by solving the GEVP for the sake of error computation, denoted as `EMP` (Exact computation Multi-way Partition). Then we can obtain the clusters generated by $\mathbf{U}'_{t=1}$ and $\mathbf{U}_{t=1}$ and the cost function is evaluated. Finally, we repeat the steps at $t = 2, 3, ...$ and obtain the approximated generalized eigenvectors, the updated clusters, and the clusters by exact solving the GEVP and Recursive Bisection.

By comparing the `UMP` and `WMP`, the error reduction performance can be shown. And the comparison between cost function value derived by `UMP` and `WMP` demonstrates the necessity of performing the update procedure. How good the `UMP` behaves can be illustrated by comparing the cost function value derived by `UMP`

and WMP. And the efficiency of the proposed updating method can be demonstrated by recording the time consumption to derive the clusters of each method. All of the comparisons will be given in the numerical results.

In the simulations, ϵ is 10^{-4} and I is 2. Within the conjugate gradient descent, the i_{max} is set to be $6\sqrt{N}$. During the updating, apart from the leading $K = 3$ components, one more generalized eigenvector is tracked as well.

7.2.2 Simulation of users' movements

The random movement of the user is achieved by randomly moving the node to another position within a square centered at the origin position. The side length of the square is 200 *meters*, identified as the *high mobility* case. In this scenario, within each variation, 20 users change their positions. Given that there are 1200 users in the network, a series of perturbations of the adjacency matrix $\{\Delta\mathbf{W}_t\}_{t=1,2,\ldots,60}$ can be determined. We gradually update the $\mathbf{U}'$ and correspondingly the clusters.

The cellular network after users' movement and the final clusters at $t = 60$ are given in Fig. 7.13. The tracking results are illustrated in Fig. 7.14 and 7.15. We illustrate the tracking error and cost function value at each time instance, where we can see how the update procedure performs when the network changes over time.

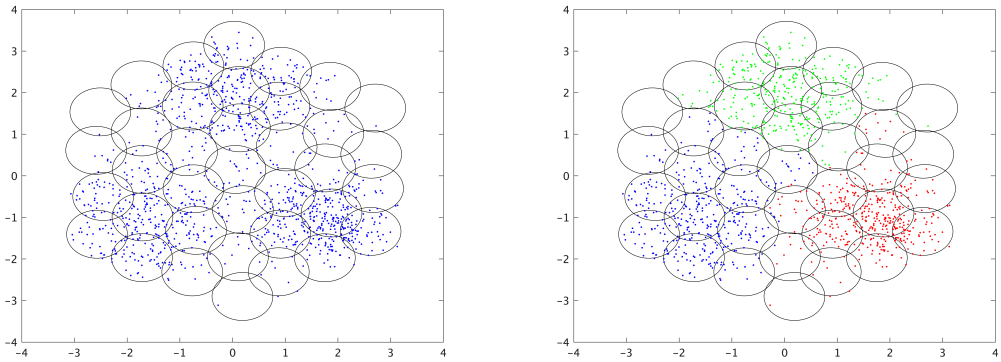


Figure 7.13: Left: Cellular network after users randomly move. Right: Clusters generated by updating the perturbed original network.

7.2.3 Simulation of new users

In this scenario, we add 100 users to each hotspot, drawn from the same distribution which is used to generate the initial graph, shown in Fig. 7.16. Therefore, we have 300 new users in total. Furthermore, each perturbation is defined as the appearances of 10 new users such that we have a series of perturbations of the adjacency matrix $\{\Delta\mathbf{W}_t\}_{t=1,2,\ldots,30}$. We would like to clarify that in this case, the perturbed adjacency matrix will be larger than the unperturbed one in size since the number of nodes increases. One should turn to the dimension-varying update explained in Section 6.2.1.1. The tracking results are illustrated in Fig. 7.17 and 7.18.

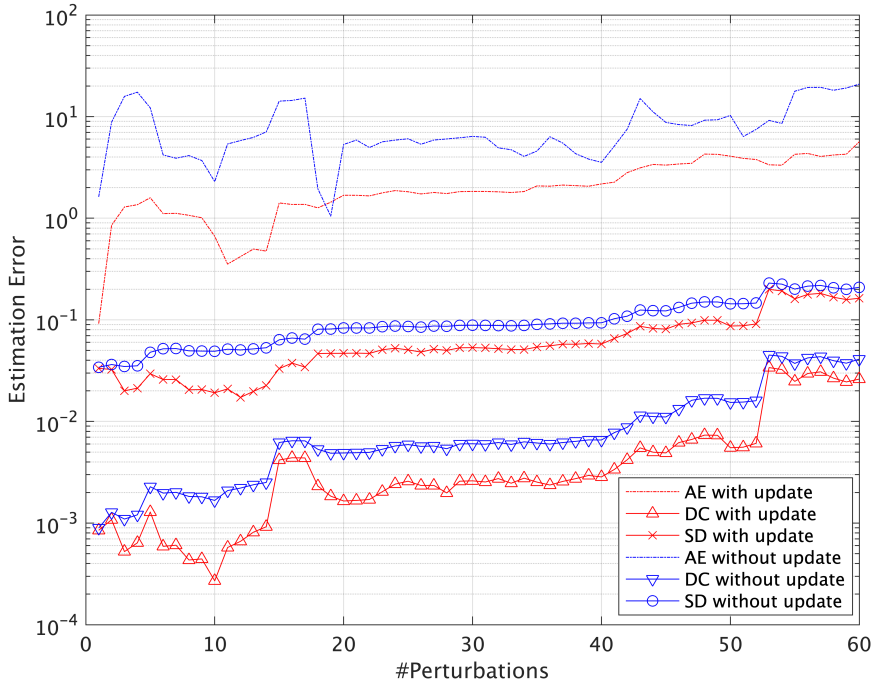


Figure 7.14: Tracking error when the graph is perturbed by users' movements.

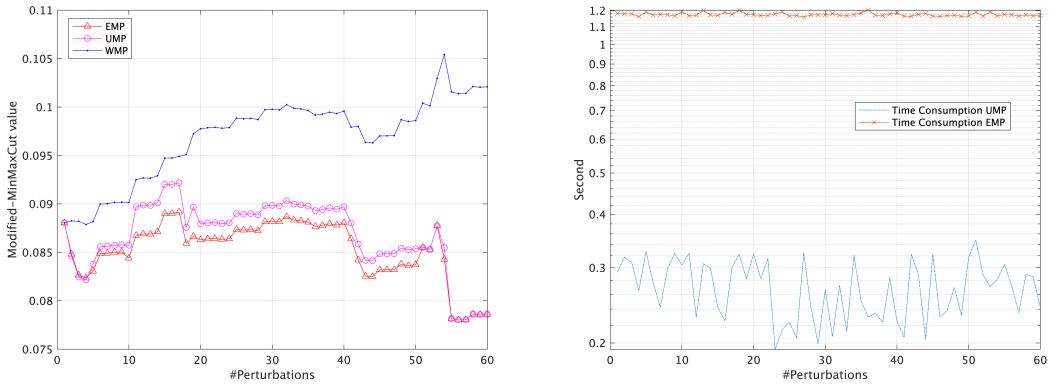


Figure 7.15: Left: *Modified-MinMaxCut* value varying with the graph perturbations by users' movement. Right: Time consumption of deriving the clusters after each perturbation.

7.2.4 Discussions

By simulating our proposed methods on the dynamic cellular network-based graph, we can observe that errors are indeed reduced compared with *without update* case. As we have shown in Fig. 7.14 and 7.17, after performing the update procedure, most of the time we can track the eigencomponents with better accuracy and less error. Even though sometimes we cannot obtain too much gain, e.g., in the final part of Fig. 7.14, it is still necessary to do so., since the cost function value is reduced compared with the *without update* case and this gain in the error-reduction

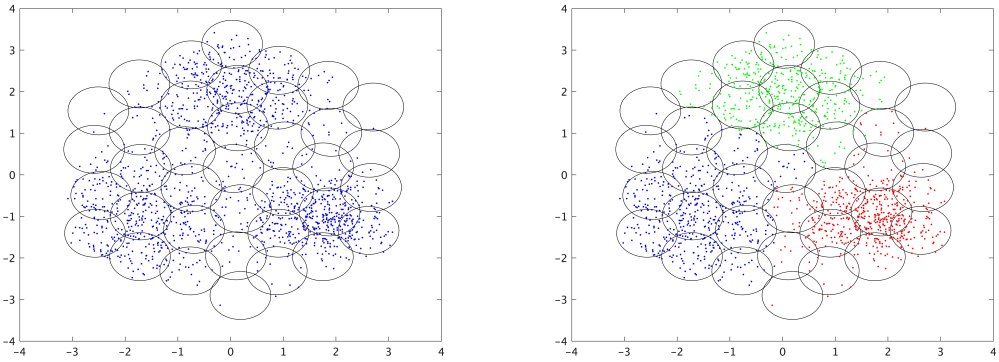


Figure 7.16: Left: Cellular network after adding new users. Right: Clusters generated by updating the perturbed original network.

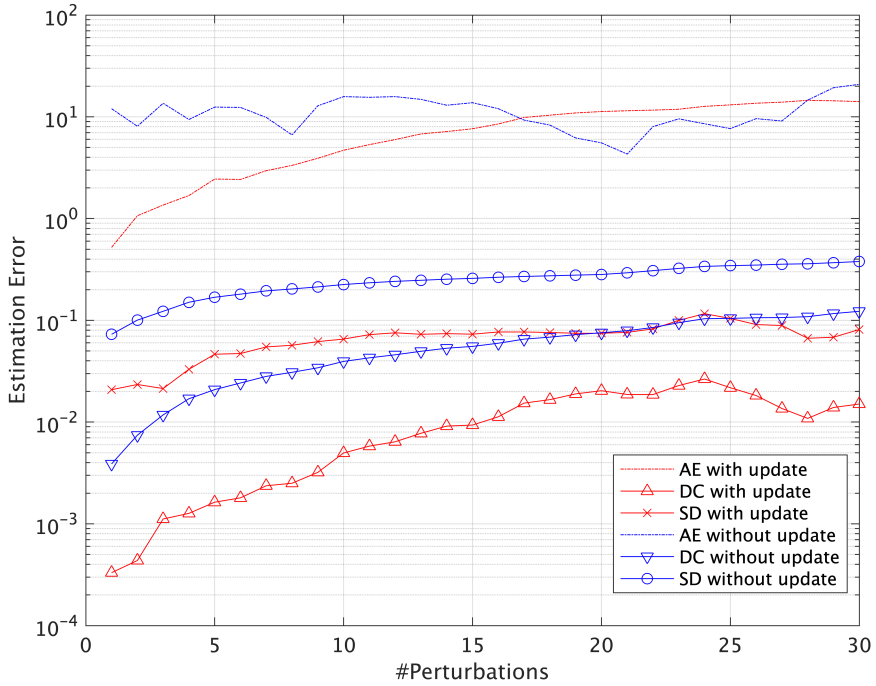


Figure 7.17: Tracking error when the graph is perturbed by new users.

enables us to have the same clusters and cost generated by exact computing the generalized eigenvectors. In terms of the cost function and time consumption, we can see that in both scenarios, performing clustering using the estimated generalized eigenvectors achieves almost the same cost values as performing clustering using the true ones, yet with less time consumption. The large cost function value in Fig. 7.18 is because we randomly assign the new users to different clusters since in this case (*without update*), without no extra effort, we have no information about to which cluster the new user should belong. And to make a fair comparison, we should not introduce extra efforts to determine the cluster, since we do not have this step in *with update* case. To sum up, the simulations illustrate

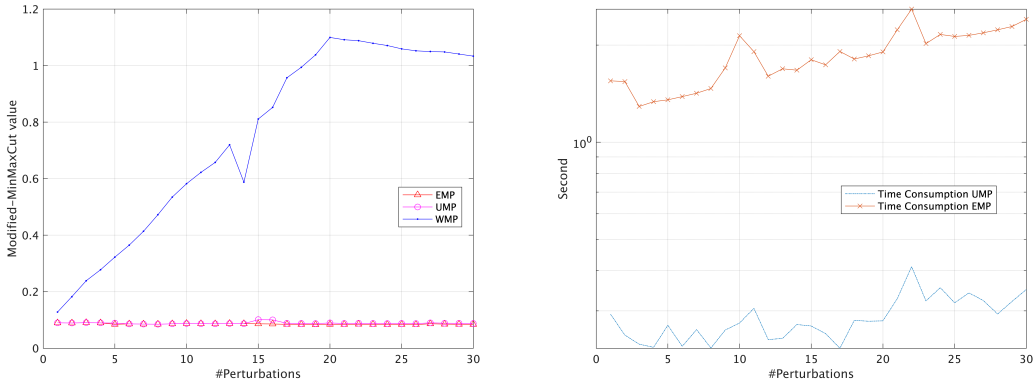


Figure 7.18: Left: *Modified-MinMaxCut* value varying with the graph perturbations by new users. Right: Time consumption of deriving the clusters after each perturbation.

the efficacy of the proposed tracking method on efficiently updating the clusters of a dynamic graph.

Bibliography

[1] A. Ghosh and R. Ratasuk, *Essentials of lte and lte-a*. Cambridge University Press, 2011.

8 Conclusion and Future Work

8.1 Conclusion

We studied two problems from the cellular network background. We needed to partition graphs that have certain structures, and to cluster their nodes to minimize certain cost functions. Furthermore, when the graph is changing over time, the clusters should be updated as well. In the first problem, we partitioned a bipartite graph by minimizing the so-called *MinMaxCut* cost function, while in the second problem, we partitioned a general graph minimizing the so-called *Modified-MinMaxCut* cost function. The solutions we proposed are under the framework of spectral clustering, where one relies on the eigenvectors of the graph matrices, e.g., the Laplacian matrix or the adjacency matrix, to partition nodes into disjoint clusters.

The spectral clustering-based relaxation of the first cost function has been widely discussed. Therefore, we mainly focused on how to partition a bipartite graph changing over time efficiently. We assumed that the number of nodes in $\mathcal{B}$ is much smaller than the number of nodes in $\mathcal{U}$. To perform the partition, one can rely on the SVD of the normalized $\tilde{\mathbf{B}}$ matrix to derive the continuous indicator for the clusters. Based on this idea, the spectral embedding of the nodes in $\mathcal{B}$ can be constructed by the left singular vectors while the spectral embedding of the nodes in $\mathcal{U}$ can be constructed by the right singular vectors, which suggest that they can be partitioned separately. Therefore, we first partitioned the nodes in $\mathcal{B}$. To partition nodes in $\mathcal{U}$, we proposed a heuristic method called *user assignment method*. In this heuristic, a node in $\mathcal{U}$ belongs to the cluster which the sum of the weight of all the edges between this node and the nodes in the cluster is the largest. Regarding the dynamic bipartite graph partition problem, we first transformed the variations in the graph structure to the perturbations of the graph matrices. Then this problem is solved by updating the left singular vectors to update the clusters of $\mathcal{B}$ and then assign users. The update of the left singular vectors can be done by the update algorithm we proposed, which is based on matrix perturbation. This method can be applied when only a few singular vectors or eigenvectors require to be updated. Our simulations show that we can partition the bipartite graph effectively.

In the second problem, we considered a new cost function, called *Modified-MinMaxCut*. Before going into the multi-way partition problem, we first considered the bisection problem. Since the problem is NP hard, it is difficult to solve the original problem directly, thus we first relaxed the binary indicator vector and allowed the entries take continuous values, called continuous indicator vectors. After that, we considered the inverse cost function, which is to be maximized. In total, we proposed two ways to relax the cost. In the first way, in order to relax the problem in a meaningful way and to simplify the optimization, we adopted

a constraint on the indicator vector. Then by dropping the binary constraint, the optimal continuous indicator can be identified by setting the derivative of the Lagrangian to $\mathbf{0}$. This leads us to a GEVP. We explained the relationship between the generalized eigenvector of the GEVP and the continuous indicator vector for bisection problem and it turned out to be the leading generalized eigenvector. Finally, to retrieve the binary indicator vector, one should set a threshold to discretize the continuous vector. The optimal way to do this is to perform line search over the threshold. The proposed methods can be extended to multi-way partition easily, by taking more eigenvectors and then performing K-means.

The second relaxation is based on rewriting the linear term into a quadratic term which can be absorbed by other quadratic terms. By doing so, we can get rid of the linear term. Then similarly, we adopted the same constraint and set the derivative of the Lagrangian to $\mathbf{0}$ and the continuous indicator can be obtained by solving another GEVP. However, for the original cost, we were not able to derive the similar conclusion. Therefore, in the second relaxation, turning to the inverse cost is the only feasible way. The multi-way partition can be achieved by taking more eigenvectors. Usually, K eigenvectors are sufficient for a K-way partition problem. Regarding partitioning graphs changing over time, we can follow the update algorithm implemented in the first problem, because in this problem, updating the clusters is equal to updating the eigencomponents. Our simulations showed that the second relaxation is less satisfactory than the first one and we can partition the graph effectively (lower cost function value) and efficiently (lower time consumption) using the first method and the update method. This concludes parts in the thesis in terms of the second problem.

8.2 Future Work

- As in the works dealing with other cost functions, the bisection problem is the first to be considered, since both the indicator vectors and the cost function can be simplified and easy to tackle. The multiway partition problem tends to be tackled later. For the cost functions, such as *MinMaxCut* [1], and *Normalized Cut* [2], the solutions towards multiway partition problem have been directly derived [3–7]. However, in this thesis work, the solution towards the multiway partition is based on the heuristic from the bisection. Therefore, the potential future work is to work on the multiway partition problem directly and approaching the multilway partition problem from other perspectives.

- The background of this thesis work is partitioning cellular networks. Although our testing graphs are generated from cellular network models, it would be better to implement the methods on the data from the real world.

- One of the assumptions in this thesis work is the number of clusters is known. However, determining the number of clusters in a graph is not a trivial problem. Therefore, another potential future work is to design algorithms that can automatically monitor the number of clusters of a dynamic cellular network such that meaningful clusters can be generated.

Bibliography

[1] C. H. Ding, X. He, H. Zha, M. Gu, and H. D. Simon, "A min-max cut algorithm for graph partitioning and data clustering," in *Proceedings 2001 IEEE international conference on data mining*. IEEE, 2001, pp. 107–114.

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

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

[4] F. Nie, C. Ding, D. Luo, and H. Huang, "Improved minmax cut graph clustering with nonnegative relaxation," in *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*. Springer, 2010, pp. 451–466.

[5] M. Gu, H. Zha, C. Ding, X. He, H. Simon, and J. Xia, "Spectral relaxation models and structure analysis for k-way graph clustering and bi-clustering," 2001.

[6] E. P. Xing and M. I. Jordan, "On semidefinite relaxations for normalized k-cut and connections to spectral clustering," 2003.

[7] T. Li and C. Ding, "The relationships among various nonnegative matrix factorization methods for clustering," in *Sixth International Conference on Data Mining (ICDM'06)*. IEEE, 2006, pp. 362–371.

Conjugate Gradient Descent

A

The conjugate gradient descent algorithm solving Equation (A.0.1) is summarized in Algorithm 10.

$$(\tilde{\mathbf{K}}^\top \tilde{\mathbf{K}} + \epsilon \mathbf{I})\Delta \mathbf{c}_m = \tilde{\mathbf{K}}^\top \tilde{\mathbf{h}} \tag{A.0.1}$$

Algorithm 10: Conjugate Gradient Descent

Data: $\tilde{\mathbf{K}}$, $\tilde{\mathbf{h}}$, ϵ
Result: $\Delta \mathbf{c}_m$

$\mathbf{A} = \tilde{\mathbf{K}}^\top \tilde{\mathbf{K}} + \epsilon \mathbf{I}$
$\mathbf{b} = \tilde{\mathbf{K}}^\top \tilde{\mathbf{h}}$
$i = 0$
$\Delta \mathbf{c}_m = \mathbf{0}$
$\mathbf{r} = \mathbf{b} - \mathbf{A}\Delta \mathbf{c}_m$
$\mathbf{d} = \mathbf{r}$
$\delta_{new} = \mathbf{r}^\top \mathbf{r}$
$\delta_0 = \delta_{new}$
while $i < i_{max}$ *and* $\delta_{new} > \epsilon^2 \delta_0$ **do**
    $\mathbf{q} = \mathbf{A}\mathbf{d}$
    $\alpha = \frac{\delta_{new}}{\mathbf{d}^\top \mathbf{q}}$
    $\Delta \mathbf{c}_m = \Delta \mathbf{c}_m + \alpha \mathbf{d}$
    $\mathbf{r} = \mathbf{r} - \alpha \mathbf{q}$
    $\delta_{old} = \delta_{new}$
    $\delta_{new} = \mathbf{r}^\top \mathbf{r}$
    $\beta = \frac{\delta_{new}}{\delta_o ld}$
    $\mathbf{d} = \mathbf{r} + \beta \mathbf{d}$
    $i = i + 1$
end

Proofs

B

B.1 Proof Proposition 7 and 9

Let us rewrite the expression (4.2.18) using the introduced variables, i.e.,

$$J_2(\mathbf{c}) = \frac{a}{b_1} + \frac{a}{b_2} = \frac{a(b_1+b_2)}{b_1 b_2}, \tag{B.1.1}$$

where the dependency on $\mathbf{c}$ has been dropped for sake of notation.

Now, defining

$$\boldsymbol{l}_1 = \partial a/\partial \mathbf{c};\ \boldsymbol{l}_2 = \partial b_1/\partial \mathbf{c};\ \boldsymbol{l}_3 = \partial b_2/\partial \mathbf{c};\ \boldsymbol{l}_4 = \partial(\mathbf{q}^\top \mathbf{c})/\partial \mathbf{c}$$

we can write

$$\frac{\partial J_2}{\partial \mathbf{c}} = \frac{\boldsymbol{l}_1 b_1 - \boldsymbol{l}_2 a}{b_1^2} + \frac{\boldsymbol{l}_1 b_2 - \boldsymbol{l}_3 a}{b_2^2}, \tag{B.1.2}$$

which when setting $\partial f/\partial \mathbf{c} = 0$ leads to

$$\boldsymbol{l}_1 = \frac{a(b_1^2+b_2^2)}{b_1 b_2 (b_1+b_2)}\boldsymbol{l}_2 + \frac{a b_1^2}{b_1 b_2 (b_1+b_2)}\boldsymbol{l}_4, \tag{B.1.3}$$

where we have used the fact that $\boldsymbol{l}_3 = \boldsymbol{l}_2 + \boldsymbol{l}_4$. Using the above expression, and substituting the corresponding values, we retrieve the expression of the result for Proposition 7. Regarding the value of ν', we can follow the way by equating two definitions of λ'.

Follow the same procedure, we can obtain

$$\boldsymbol{l}_4 = -\frac{a b_1^2}{b_1 b_2 (b_1+b_2)} 2\mathrm{diag}(\mathbf{q})\mathbf{c}_m, \tag{B.1.4}$$

consider that

$$J_2(\mathbf{c}) = \frac{a}{b_1} + \frac{a}{b_2} = \frac{a(b_1+b_2)}{b_1 b_2}, \tag{B.1.5}$$

we can multiply one more (b_1+b_2) on both numerator and denominator, constructing a $J_2(\mathbf{c})$ term. Rewriting the equation and finally we can obtain the result for Proposition 9.

B.2 Proof of the Upper bound

We have

$$\lambda' = \frac{4b_1 b_2 (b_1+b_2+\nu' b_1 b_2)}{a(b_1^2+b_2^2)} = \frac{4(b_1+b_2)^2}{J(b_1^2+b_2^2)} + \frac{4\nu' b_1^2 b_2^2}{b_1^2+b_2^2}.$$

Let's look into the first term. With the Cauchy–Schwarz inequality

$$(u_1v_1 + u_2v_2)^2 \leq (u_1^2 + u_2^2)(v_1^2 + v_2^2),$$

and setting $u_1 = b_1$, $u_2 = b_2$, and $v_1 = v_2 = 1$, we obtain

$$(b_1 + b_2)^2 \leq 2(b_1^2 + b_2^2).$$

Regarding the second term, we plug $\nu' = -\kappa(b_1^2 + b_2^2)^{-1}$ in where $\kappa = \sum_{i\in\mathcal{B},j\in\mathcal{U}} w_{i,j} = \mathbf{1}^\top\tilde{\mathbf{W}}\mathbf{1}$ is a positive constant value for a certain graph, we can then obtain

$$\lambda' \leq \frac{8}{J_2} - 4\kappa\big(\frac{b_1b_2}{b_1^2 + b_2^2}\big)^2 < \frac{8}{J_2},$$

since the second term is a positive value. By exchanging positions of λ' and J_2, the upper bound (4.2.20) can be obtained. Here since either b_1 or b_2 is non-negative, the equality cannot thus be obtained.

B.3 Proof of Negative Semidefinite

In the relaxation method 2, we introduce a matrix $\mathbf{W}_{\mathrm{DS}}$ defined as follows

$$\begin{aligned}\mathbf{W}_{\mathrm{DS}} &= \mathbf{W}_{\mathrm{D}}^\top + \mathbf{W}_{\mathrm{D}}, &\text{(B.3.1)}\\ &= \bar{\mathbf{W}}^\top + \bar{\mathbf{W}} - 2\mathrm{diag}(\mathbf{q}), &\text{(B.3.2)}\\ &= 2(\tilde{\mathbf{W}}^\top + \tilde{\mathbf{W}} - \mathrm{diag}(\mathbf{q})). &\text{(B.3.3)}\end{aligned}$$

Recall the definition of $\mathbf{q} = (\tilde{\mathbf{W}}^\top + \tilde{\mathbf{W}})\mathbf{1}$, we have

$$\mathbf{W}_{\mathrm{DS}} = 2(\tilde{\mathbf{W}}^\top + \tilde{\mathbf{W}} - \mathrm{diag}((\tilde{\mathbf{W}}^\top + \tilde{\mathbf{W}})\mathbf{1})). \tag{B.3.4}$$

For the sake of convenience, we denote $\tilde{\mathbf{W}}^\top + \tilde{\mathbf{W}}$ as $\mathbf{A}$ and ignore the constant coefficient. Then we would like to prove the matrix $\mathbf{A} - \mathrm{diag}(\mathbf{A}\mathbf{1})$ to be negative semidefinite.

Consider the following symmetric matrix $\mathbf{A} \in \mathbb{R}^{N\times N}$, whose entries are non-negative

$$\mathbf{A} = \begin{bmatrix} a_{11} & a_{12} & \dots & a_{1N} \\ a_{21} & a_{22} & \dots & a_{2N} \\ \vdots & \vdots & \ddots & \vdots \\ a_{N1} & a_{N2} & \dots & a_{NN} \end{bmatrix}, \tag{B.3.5}$$

the matrix $\mathrm{diag}(\mathbf{A}\mathbf{1})$ can be rewritten as

$$\mathrm{diag}(\mathbf{A}\mathbf{1}) = \begin{bmatrix} \sum_{i=1}^N a_{1i} & 0 & \dots & 0 \\ 0 & \sum_{i=1}^N a_{2i} & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & \sum_{i=1}^N a_{Ni} \end{bmatrix}. \tag{B.3.6}$$

Then we have

$$\mathbf{A} - \text{diag}(\mathbf{A1}) \tag{B.3.7}$$

$$= \begin{bmatrix} a_{11} & a_{12} & \dots & a_{1N} \\ a_{21} & a_{22} & \dots & a_{2N} \\ \vdots & \vdots & \ddots & \vdots \\ a_{N1} & a_{N2} & \dots & a_{NN} \end{bmatrix} - \begin{bmatrix} \sum_{i=1}^{N} a_{1i} & 0 & \dots & 0 \\ 0 & \sum_{i=1}^{N} a_{2i} & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & \sum_{i=1}^{N} a_{Ni} \end{bmatrix} \tag{B.3.8}$$

$$= \begin{bmatrix} -\sum_{i=2}^{N} a_{1i} & a_{12} & \dots & a_{1N} \\ a_{21} & -\sum_{i=1, i\neq 2}^{N} a_{2i} & \dots & a_{2N} \\ \vdots & \vdots & \ddots & \vdots \\ a_{N1} & a_{N2} & \dots & -\sum_{i=1}^{N-1} a_{Ni} \end{bmatrix}. \tag{B.3.9}$$

Theorem 1. Gershgorin circle theorem. Let $\mathbf{A}$ be an $N \times N$ matrix, with entries a_{ij}. For $i \in \{1, ..., N\}$ let R_i be the sum of the absolute values of the non-diagonal entries in the i-th row

$$R_i = \sum_{j \neq i} |a_{ij}|. \tag{B.3.10}$$

Let $D(a_{ii}, R_i)$ be a closed disc centered at a_{ii} with radius R_i. Such a disc is called a Gershgorin disc. Then every eigenvalue of $\mathbf{A}$ lies within at least one of the Gershgorin discs $D(a_{ii}, R_i)$.

By directly applying the Gershgorin circle theorem, we can find that all the eigenvalues of our $\mathbf{A} - \text{diag}(\mathbf{A1})$ matrix should be less than or equal to 0, meaning that the matrix is negative semidefinite.