Finding Representative Sampling Subsets on Graphs: Leveraging Submodularity

by

Tianyi Li

Student Name	Student Number
Tianyi Li	5464897

Instructor: prof.dr.ir.Geert Leus
Project Duration: November, 2022 - July, 2023
Faculty: EEMCS

TU Delft

Acknowledgments

I want to thank my supervisor Prof. Geert Leus whose door was always open for me when I needed help. Talking to him was always inspiring.

I would also like to thank my friends for the company during the year. Spending time with them can always make me forget about all the troubles.

Master of Science would not have been possible without my family's support. I am thankful to my parents, who have been with me almost every step of my life.

Abstract

In this work, we deal with the problem of reconstructing a complete bandlimited graph signal from partially sampled noisy measurements. For a known graph structure, some efficient centralized algorithms are proposed to partition the nodes of the graph into disjoint subsets such that sampling the graph signal from any of these subsets leads to a sufficiently accurate reconstruction. Furthermore, we consider the situation when the graph has a massive size, where processing the data centrally is impractical anymore. To overcome this issue, a distributed framework is proposed that allows us to implement the centralized algorithms in a parallelized fashion. Finally, we provide numerical simulation results on synthetic and real-world data to show that our proposals outperform state-of-the-art node partitioning techniques.

Contents

Nomenclature

Graph

W	Graph weighted matrix
A	Graph adjacency matrix
L	Graph Laplacian matrix
$\mathcal{G}(\mathcal{V}, \mathcal{E})$	Graph $\mathcal{G}$ with node set $\mathcal{V}$ and edge set $\mathcal{E}$

Linear Algebra

$(\cdot)^\top$	Transpose
$(\cdot)^{-1}$	Inverse
$(\cdot)^\dagger$	Inverse
$\mathbb{R}^n$	Real vector space of n-dimensional vectors
$\mathbb{R}^{n \times n}$	Real matrices space of n by n matrices
I	Identity matrix
1	All-one vector
diag{**x**}	Diagonal matrix with **x** on the diagonal
tr{**X**}	Trace of a matrix

Mathematical Object

x	Scalar
x	Vector
X	Matrix
$[\mathbf{x}]_i$	The i-th entry of a vector
$\mathbf{x}_K$	The first K entries of a vector
$\mathbf{x}_\mathcal{S}$	The coefficients of a vector whoses indices are in set S
$[\mathbf{X}]_{i,j}$	The (i,j)-th element of a matrix
$[\mathbf{X}]_{i,:}$	The i-th row of a matrix
$[\mathbf{X}]_{:,i}$	The i-th column of a matrix
$\mathbf{X}_K$	First K columns of a matrix
$\mathcal{S}(i)$	The i-th element of set $\mathcal{S}$
| **x** |	Cardinality of vector **x**

Set

$\mathcal{V}$	Set

$2^{\mathcal{V}}$	Power set of set V
$\mid \mathcal{V} \mid$	Cardinality of set $\mathcal{V}$
$\in$	Belongs to (element-wise)
$\subseteq$	Belongs to (set-wise)
$\cup$	Union
$\cap$	Intersection
$\varnothing$	Empty set

List of Figures

List of Tables

1 Introduction

A graph consisting of nodes and edges is a helpful data representation when some topology structures exist in the data domain. The data on the graph node can be seen as a finite collection of samples living on the graph as the domain. This signal-graph coupling structure is a graph signal [1].

In the emerging field of graph signal processing (GSP), roughly speaking, two essential topics are being studied. In the first, we assume that the underlying graph structure is given and extend concepts from classical signal processing, like filtering [2], sampling and interpolation [3], [4], and signal recovery [5] to process graph signals. The second topic considers the case where the graph structure is unknown, and from the receiving data, the goal is to learn the topology of graphs. Related works could be found in [6], [7]. In this work, we focus on the first topic, to be more specific, the graph sampling theory.

Applications of GSP arise very often. Here are some examples. In transportation networks [1], we may be interested in analyzing epidemiological data which suggests the spread of disease. In brain networks, we are interested in finding ways of detecting the anomaly patterns that appear in the cerebral cortex [8]. In movie recommendation systems, it is possible to predict users' ratings from similar users, i,e. their neighbors.

1.1. Problem Statement

In our study, we consider as graph an Internet of Things (IoT) sensor network, and the graph signal some environmental characteristics measured by the sensors, such as temperature, humidity, etc. Due to the massive size of many real-world sensor networks, observing all the sensor data in a graph is often challenging. Moreover, sampling data on every sensor could be highly energy-consuming. To solve this issue, one could sample a subset of the network and reconstruct the whole graph signal based on that observation. Therefore, the objective is to partition the sensor nodes into several disjoint subsets so that each sensor node can only be selected in one subset. In each sampling round, only a subset of sensors is used to reconstruct the graph signal. Meanwhile, ensuring each subset's reconstruction error is sufficiently small is essential. Suppose we partition the graph into P disjoint subsets and sample every subset in turn. In that case, each sensor has to transmit P times less frequently, reducing energy consumption and extending the sensor's life span by P times.

At first glance, this partitioning task looks similar to spectral clustering [9], [10], which also aims to partition the graph into disjoint subsets. But to solve our problem, finding a partition based on spectral clustering makes no sense, where nodes are clustered based on their similarity, i.e., nodes within a cluster are similar. However, reconstructing the complete graph signal using a set of similar observations is never a good idea. The reason for this is that this will introduce too much redundancy. Instead, intuitively speaking, we should put dissimilar nodes into a subset. Indeed, these are just some insights into what the solution should look like. The formal problem formulation will be introduced in Chapter 4.

1.2. Outline

The thesis is organized as follows:

- Chapter 2-Background: We give the background related to this work, including some graph signal processing concepts and the basics of submodular maximization
- Chapter 3-Greedy Graph Node Selection: We introduce the single subset selection problem as our partitioning problem is based on that.
- Chapter 4-Related Work and Problem Formulation: We review the existing approaches for the partitioning problem and explain why they are not good enough. Then, we formally formulate the problem.
- Chapter 5-Partitioning Algorithms: New algorithms are provided in this chapter to tackle the problem defined in Chapter 4.
- Chapter 6-Distributed Partitioning Framework: We propose a distributed framework that allows us to implement the algorithms from the previous chapters in parallel.
- Chapter 7-Results and Discussion: We show some results of our algorithms and give guidance about how to use them in different scenarios.

2 Background

This chapter introduces two critical topics essential for the rest of this report: graph signal processing and submodular maximization.

2.1. Graph Signal Processing

In this section, we go through some concepts in graph signal processing. The relevant theory generalizes classical signal processing from regular to irregular domains commonly described by graphs [3].

2.1.1. Graph Representations

Before going to the graph signal, we briefly introduce different graph representation methods. In mathematics, a graph $\mathcal{G} = (\mathcal{N}, \mathcal{E})$ is defined as a finite set of nodes $\mathcal{N} = \{v_1, v_2, \ldots, v_N\}$ and a limited set of edges $\mathcal{E}$ [11], which is a generic data representation form that is useful for depicting the topological structure of data domains in various applications, including social networks, transportation networks(see Fig. 7.2), sensor networks, and brain networks, etc [1]. The weight associated with an edge usually implies the similarity between the two vertices it connects, i.e., a high weight suggests that the corresponding nodes are pretty similar. In contrast, a zero weight means that the pair of nodes are not connected.

In some applications, a graph structure is not explicitly given, so it must first be constructed from the data. For a sensor network, this can be done by connecting every k nearest neighbors and setting the weights to be

$$[\mathbf{W}]_{m,n} = \frac{e^{-d_{n,m}^2}}{\sqrt{\sum_{k\in\mathcal{N}_n} e^{-d_{n,k}^2} \sum_{\ell\in\mathcal{N}_m} e^{-d_{m\ell}^2}}}, \tag{2.1}$$

where $[\mathbf{W}]_{m,n}$ denotes the weight between node n and node m, $\mathcal{N}_n$ denotes the neighbors of node n and $d_{n,m}$ denotes the geodesical distance between the nth and mth nodes [12]. Other graph construction methods can be found in [13].

Algebraic matrices, for example, the adjacency matrix and the Laplacian matrix, can represent graphs. In some literature, one can also use the incidence matrix to represent graphs, but here we focus on the first two types of matrices. The (weighted) adjacency matrix of a graph $\mathcal{G} = (\mathcal{N}, \mathcal{E})$ is simply given by

$$[\mathbf{A}]_{u,v} = \begin{cases} [\mathbf{W}]_{u,v}, & \text{if } (u,v) \in \mathcal{E} \\ 0, & \text{if } (u,v) \notin \mathcal{E}. \end{cases} \tag{2.2}$$

For a directed graph, $(u, v) \in \mathcal{E}$ does not always imply that $(v, u) \in \mathcal{E}$, so its adjacency matrix is generally asymmetric. We assume that all the graphs mentioned in this report are undirected, i.e., their corresponding adjacency matrices are symmetric. We can also introduce the Laplacian matrix for such undirected as done next.

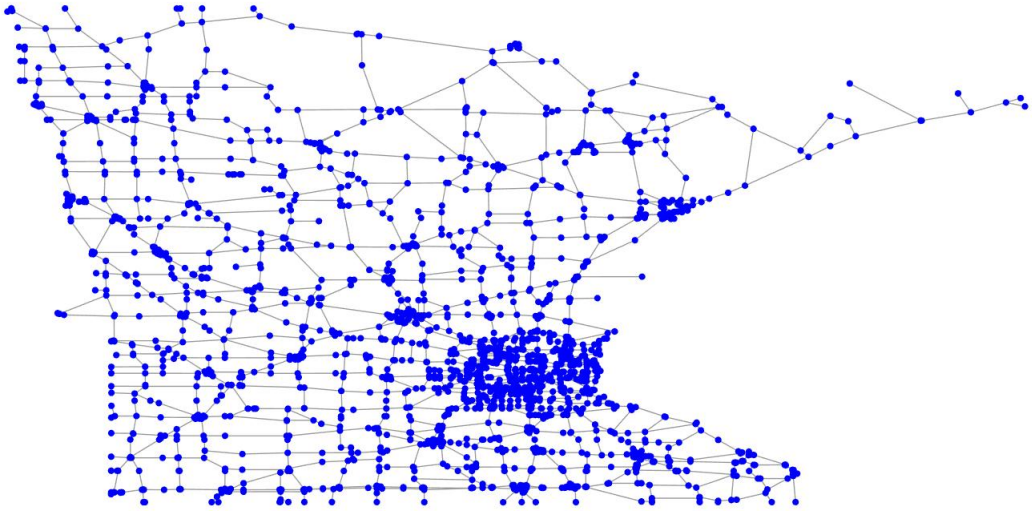

Figure 2.1: An example of a graph: the Minnesota road network with 2642 nodes generated by the GSPBOX toolbox [14]

.

Before introducing the Laplacian matrix, we define the degree matrix $\mathbf{D} = \text{diag}\,(d_1, \ldots, d_N)$, where

$$d_u := \sum_{v=1}^{N} [\mathbf{A}]_{u,v}. \tag{2.3}$$

Once we obtain the adjacency matrix and the degree matrix of a graph, its Laplacian matrix is then given by

$$\mathbf{L} = \mathbf{D} - \mathbf{A} \tag{2.4}$$

2.1.2. Graph Spectrum

Spectral graph theory studies the eigenvalues and eigenvectors of matrices associated with graphs [9]. A common choice of such a matrix for undirected graphs is the graph Laplacian matrix. It can be decomposed as

$$\mathbf{L} = \mathbf{U}\mathbf{\Lambda}\mathbf{U}^\top, \tag{2.5}$$

where $\mathbf{\Lambda} = \text{diag}\,(\lambda_1, \lambda_2, \ldots, \lambda_N)$ is a diagonal matrix collecting the eigenvalues λ_n and $\mathbf{U} = [\mathbf{u}_1, \mathbf{u}_2, \ldots, \mathbf{u}_N]$ is the matrix collecting the eigenvectors $\mathbf{u}_n$. Without loss of generality, we assume these eigenvalues are placed in increasing order for convenience. Notice that the smallest eigenvalue $\lambda_1 = 0$, and the corresponding eigenvector is given by $\mathbf{u}_1 = [1, 1, \ldots, 1]^\top / \sqrt{N} = \mathbf{1}/\sqrt{N}$.

Clustering is a fundamental technique broadly used in data mining, unsupervised learning, etc. Given the graph, it is possible to do clustering based on its spectrum.

A K-means based spectral clustering algorithm is given in Algorithm 1 [15], [16]. For more details of the K-means algorithm, see [17].

We will see later that a preprocessing procedure like clustering might help to improve the efficiency of our node partitioning problem.

2.1.3. Graph Signals

By assigning a scalar value to each graph node, we can define a signal on top of the graph, called a graph signal. Note that in more general cases, we can map multiple values or a vector to a single

Algorithm 1 Spectral Clustering

Input: Graph Laplacian **L**, number of clusters K
Output: K clusters
1: Compute the eigenvalue decomposition of **L**
2: Choose the first K eigenvectors corresponding to K smallest eigenvalues and form the $N \times K$ matrix $\mathbf{U}_K$
3: Assign the ith node to the ith row of $\mathbf{U}_K$
4: Run K-means on the rows of $\mathbf{U}_K$

node, and the corresponding graph signal is then multivariate. In this paper, we restrict the node-signal mapping to one-to-one. Once the order is fixed, we can stack the signal values into a vector

$$\mathbf{x} = [x_1, x_2, \ldots, x_N]^\top, \tag{2.6}$$

where the ith value corresponds to node i. Note that the order of the nodes does not matter as long as it is consistent with the order of the nodes in the Laplacian.

Graph Fourier Transform

In traditional Fourier theory, the frequency-domain signal corresponding to a time-domain signal is the expansion of it in terms of complex exponentials [18]

$$\hat{f}(\xi) = \int_{-\infty}^{\infty} f(x) e^{-j2\pi\xi x} dx. \tag{2.7}$$

We denote the nth eigenvector of the Laplacian as $\mathbf{u}_n$, hence its ith entry is denoted as $[\mathbf{u}_n]_i$. Analogously, we can define the graph Fourier transform $\hat{\mathbf{x}}$ of the graph signal **x** in this way:

$$[\hat{\mathbf{x}}]_n = \langle \mathbf{x}, \mathbf{u}_n \rangle = \sum_{i=1}^{N} x_i [\mathbf{u}_n]_i^*. \tag{2.8}$$

Likewise, the inverse graph Fourier transform is given by

$$[\mathbf{x}]_i = \sum_{n=1}^{N} [\hat{\mathbf{x}}]_n [\mathbf{u}_n]_i. \tag{2.9}$$

In vector form, the graph Fourier transform of **x** is given by

$$\hat{\mathbf{x}} = \mathbf{U}^\top \mathbf{x}, \tag{2.10}$$

and the inverse graph Fourier transform by

$$\mathbf{x} = \mathbf{U}\hat{\mathbf{x}}. \tag{2.11}$$

In classical Fourier analysis, complex exponentials associated with low frequencies, i.e., ξ close to zero, tend to oscillate smoothly, while the complex exponentials associated with high frequencies oscillate rapidly.

In the graph frequency domain, the frequencies of a graph signal are defined by the eigenvalues of its Laplacian. Larger eigenvalues correspond to higher frequency components and vice versa. For a connected graph, as mentioned above, the eigenvector associated with the smallest eigenvalue, which equals zero, is a DC-valued vector and equals $1/\sqrt{N}$ at each node. For larger eigenvalues, the corresponding eigenvectors oscillate more rapidly over the graph. This coincides with the theory of classical Fourier analysis discussed above. We will discuss the topic of smoothness further in the following section, which complements the graph Fourier transform theory.

Graph Signal Smoothness

Let us start from the viewpoint of classical signal processing. In the time domain, a smooth signal implies that its magnitude changes slowly over time. However, in the graph domain, because of its irregularity, it is not appropriate to define the smoothness based on the order of the nodes, or else a different ordering would lead to a different result, which is incorrect. It makes much more sense to define the smoothness of a graph signal based on the graph's topological structure. To be more specific, for a graph signal, if the signal values at adjacent nodes are very similar, then this graph signal is smooth.

Mathematically, for a graph signal $\mathbf{x}$, the measure of smoothness could be formulated by the discrete $p-$Dirichlet form [1] which is defined as

$$S_p(\mathbf{x}) := \frac{1}{p}\sum_{i\in\mathcal{N}} \|\nabla_i \mathbf{x}\|_2^p = \frac{1}{p}\sum_{i\in\mathcal{N}} \left[\sum_{j\in\mathcal{N}_i} W_{ij}[x_j - x_i]^2\right]^{\frac{p}{2}}, \tag{2.12}$$

where $\|\nabla_i \mathbf{x}\|_2$ is called the local smoothness of $\mathbf{x}$ at node i, and is defined as $\|\nabla_i \mathbf{x}\|_2 := \left[\sum_{j\in\mathcal{N}_i} W_{ij}[x_j - x_i]^2\right]^{\frac{1}{2}}$. It measures the local smoothness of the graph signal near node i. We can play with p according to different requirements. When $p = 2$, then we have

$$\begin{aligned} S_2(\mathbf{x}) &= \frac{1}{2}\sum_{i\in\mathcal{N}}\sum_{j\in\mathcal{N}_i} W_{ij}[x_j - x_i]^2 \\ &= \sum_{(i,j)\in\mathcal{E}} W_{ij}[x_j - x_i]^2 = \mathbf{x}^\top \mathbf{L}\mathbf{x}. \end{aligned} \tag{2.13}$$

$S_2(\mathbf{x})$ is called the graph Laplacian quadratic form [9]. This value is small only when the signal has similar values at neighboring nodes connected by an edge with a large weight.

Now we explain why the eigenvectors of the graph Laplacian corresponding to smaller eigenvectors are more smooth. According to the Courant-Fischer theorem [19], the graph Laplacian eigenvalues and eigenvectors could also be defined as

$$\lambda_n = \min_{\substack{\mathbf{x}\in\mathbb{R}^N \\ \|\mathbf{x}\|_2=1 \\ \mathbf{x}\perp \mathrm{span}\{\mathbf{u}_1,\ldots,\mathbf{u}_{n-1}\}}} \left\{\mathbf{x}^\top \mathbf{L}\mathbf{x}\right\}, \quad n = 1, 2, \ldots, N, \tag{2.14}$$

and the minimizer is given by $\mathbf{u}_n$. From (2.13) and (2.14), we see again the strong relation between the frequency and the eigenvalue of the graph Laplacian. If we go back to (2.8) and (2.9), it is clear that the energy of a high-frequency signal concentrates on the high-frequency components, while the energy of a low-frequency signal concentrates on the low-frequency components. Figure 2.2 shows examples of high-frequency and low-frequency graph signals.

2.1.4. Graph Sampling Theory

In classical signal processing, perfect interpolation can be achieved if we sample a bandlimited signal at Nyquist rate, i.e., twice the highest frequency of a bandlimited signal. Analogously, such perfect interpolation is also possible in graph signal processing when specific requirements are satisfied. In this section, we study the sampling theory in GSP.

Broadly speaking, there are two sampling approaches in GSP. The first is *selection sampling*, in which a subset of nodes are collected and observed [3], [20]. Another approach is called *aggregation sampling*, in which the signal is observed at a single node, and we apply the graph shift operator $\mathbf{S}$ recursively [21]. In this work, only *selection sampling* is considered.

Sampling methods can be divided into two categories for *selection sampling*. The first builds on spectral graph theory, where the bandwidth of the graph signal is determined by the values of graph frequencies that correspond to the eigenvalues of the graph Laplacian matrix. The problem of finding a sampling set of a bandlimited signal boils down to finding a uniqueness set for the *Paley-Wiener* space [4], [22]. The second defines a signal's bandwidth based on its GFT's cardinality, which allows us to solve complex sampling problems by using simple linear algebra tools [3]. The thesis is based on the second, which we will discuss next. We first define what is a bandlimited graph signal.

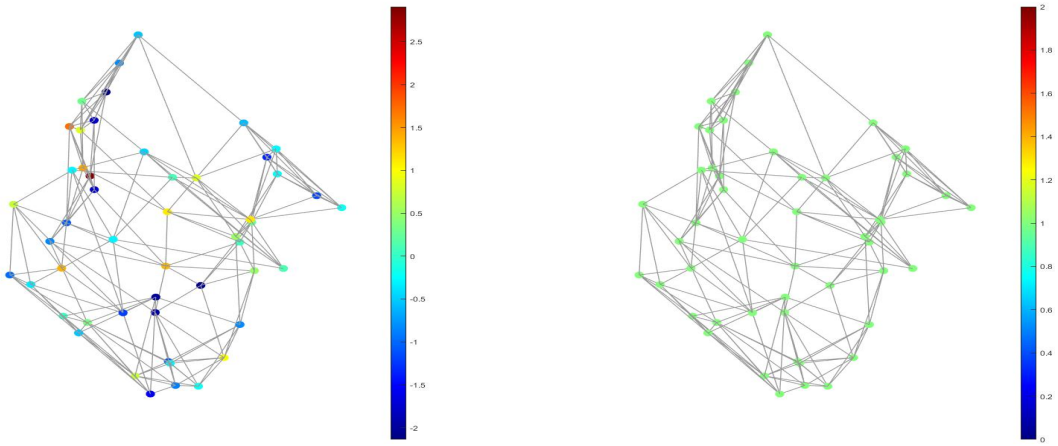

Figure 2.2: Example of a high-frequency graph signal (left) and a low-frequency graph signal (right). Different colors correspond to different signal values.

Definition 1. *A graph signal is called bandlimited if there exists a K $(K < N)$ such that its GFT satisfies*

$$[\hat{\mathbf{x}}]_k = 0 \quad \textit{for all } k > K, \tag{2.15}$$

and the smallest such K is called the bandwidth of the signal.

In the remainder of this report, the term "A K-bandlimited signal" refers to "A signal with bandwidth K." Obviously, for a K-bandlimited graph signal, $\mathbf{x} = \mathbf{U}_K \hat{\mathbf{x}}_K$, where $\mathbf{U}_K$ stacks the first K columns of $\mathbf{U}$ and $\hat{\mathbf{x}}_K$ collects the first K entries of $\hat{\mathbf{x}}$.

Suppose that we sample M nodes of a graph signal $\mathbf{x} \in \mathbb{R}^N$ to produce a sampled signal $\mathbf{x}_\mathcal{M} \in \mathbb{R}^M (M < N)$, where $\mathcal{M} = (i_1, \ldots, i_M)$ denotes the set of nodes that are sampled, and $i_m \in \{1, 2, \ldots, N\}$. We then interpolate $\mathbf{x}_\mathcal{M}$ to obtain $\mathbf{x}' \in \mathbb{R}^N$, which recovers $\mathbf{x}$ either exactly or approximately. The sampling operator $\mathbf{\Phi}$ is a linear mapping from $\mathbb{R}^N$ to $\mathbb{R}^M$, defined as

$$[\mathbf{\Phi}]_{i,j} = \begin{cases} 1, & j = i_m \\ 0, & \text{otherwise} \end{cases}, \tag{2.16}$$

and the interpolation operator $\mathbf{\Psi}$ is a linear mapping from $\mathbb{R}^M$ to $\mathbb{R}^N$. In other words, we can write

$$\text{sampling: } \mathbf{x}_\mathcal{M} = \mathbf{\Phi}\mathbf{x} \in \mathbb{R}^M,$$
$$\text{interpolation : } \mathbf{x}' = \mathbf{\Psi}\mathbf{x}_\mathcal{M} = \mathbf{\Psi}\mathbf{\Phi}\mathbf{x} \in \mathbb{R}^N.$$

We next show the sampling theorem for a bandlimited graph signal.

Theorem 1. *Let Φ satisfy*

$$\text{rank}\,(\mathbf{\Phi}\mathbf{U}_K) = K,$$

where $\mathbf{U}_K \in \mathbb{R}^{N \times K}$ denotes the first K columns of $\mathbf{U}$. For all K-bandlimited signals $\mathbf{x}$, perfect recovery, $\hat{\mathbf{x}} = \mathbf{\Psi}\mathbf{\Phi}\mathbf{x}$, is achieved by choosing

$$\mathbf{\Psi} = \mathbf{U}_K (\mathbf{\Phi}\mathbf{U}_K)^\dagger.$$

Such sampling operator is called a *qualified sampling operator*. Note that for a K-bandlimited graph signal, there might exist multiple *qualified sampling operators*. Indeed, to satisfy the full-rank condition, $M \geq K$ is a necessary condition. An interpretation is that according to Theorem 1, the sampling operator should select at least K-linearly independent rows in $\mathbf{U}_K$, and to achieve this, M should at

least be equal to K. When $M > K$, redundancy is introduced and can be exploited to increase the robustness of the sampled signal.

Given the number of samples, there is generally an optimal sampling set for noisy cases. However, this problem is generally an NP-hard problem [23]. Greedy algorithms [24] and convex optimization [25] through relaxation are the two most popular ways to deal with such problems. In the following chapters, we will discuss algorithms based on these approaches in detail.

2.2. Submodular Optimization

Submodularity is a property of set functions, which is of great significance as many functions that appear in practical problems turn out to be submodular functions, including graph cut functions [26], mutual information [27], [28], etc. The submodular function is a type of set function analogous to both convex and concave functions simultaneously. It is like a convex function in that some continuous extensions like Lovász extension [29] turn the original discrete set function into a convex function. Therefore, many submodular minimization algorithms build on its convex-like properties [30]. On the other hand, its diminishing returns property which we will introduce next, is very similar to the non-increasing derivative property of concave functions. Hence, this property is helpful in submodular maximization problems [31]. In this work, we mainly focus on its concavity point of view. Formally, a submodular function is defined as follows.

Definition 2 (Submodular function)**.** *A set function $f\colon 2^{\mathcal{N}} \to \mathbb{R}$ is a submodular function if and only if for all subsets $\mathcal{A}, \mathcal{B} \subseteq \mathcal{N}$, we have*

$$f(\mathcal{A}) + f(\mathcal{B}) \geqslant f(\mathcal{A} \cup \mathcal{B}) + f(\mathcal{A} \cap \mathcal{B}).$$

We mentioned before that submodular functions have a diminishing returns property. With this property comes an equivalent yet different definition for submodularity.

Definition 3 (Diminishing returns)**.** *The set function f is submodular if and only if for all $\mathcal{A} \subseteq \mathcal{B} \subseteq \mathcal{N}, k \notin \mathcal{B}$ and $k \in \mathcal{N}$, we have*

$$f(\mathcal{A} \cup \{k\}) - f(\mathcal{A}) \geqslant f(\mathcal{B} \cup \{k\}) - f(\mathcal{B})$$

Proof. Let $\mathcal{A} \subseteq \mathcal{B}$, and $k \notin \mathcal{B}$, we have

$$f(\mathcal{A} \cup \{k\}) - f(\mathcal{A}) - f(\mathcal{B} \cup \{k\}) + f(\mathcal{B}) = f(\mathcal{C}) + f(\mathcal{D}) - f(\mathcal{C} \cup \mathcal{D}) - f(\mathcal{C} \cap \mathcal{D}) \geqslant 0,$$

with $C = A \cup \{k\}$ and $D = B$, which implies that the condition is necessary. To show that the condition is sufficient, we firstly show that if $\mathcal{A} \subseteq \mathcal{B}$ and $\mathcal{C} \cap \mathcal{B} = \varnothing$, then

$$f(\mathcal{A} \cup \mathcal{C}) - f(\mathcal{A}) \geqslant f(\mathcal{B} \cup \mathcal{C}) - f(\mathcal{B}),$$

which can be obtained by using telescoping sums:

$$\sum_{k=1}^{m} \left(f\left(\mathcal{A} \cup \{c_1 \cdots c_k\}\right) - f\left(\mathcal{A} \cup \{c_1 \cdots c_{k-1}\}\right) \geqslant \sum_{k=1}^{m} \left(f\left(\mathcal{B} \cup \{c_1 \cdots c_k\}\right) - f\left(\mathcal{B} \cup \{c_1 \cdots c_{k-1}\}\right),\right.\right.$$

where $\mathcal{C} = \{c_1, \cdots, c_m\}$. Then, for any $\mathcal{P}, \mathcal{Q} \subseteq \mathcal{N}$, let $\mathcal{A} = \mathcal{P} \cap \mathcal{Q}$, $\mathcal{C} = P \setminus Q$ and $\mathcal{B} = \mathcal{Q}$, which is equivalent to

$$f(\mathcal{P}) + f(\mathcal{Q}) \geqslant f(\mathcal{P} \cup \mathcal{Q}) + f(\mathcal{P} \cap \mathcal{Q}),$$

and the proof finishes. □

Among all submodular functions, an important subclass is called *monotone* submodular functions.

Definition 4 (Monotonicity)**.** *A set function $f : 2^{\mathcal{N}} \to \mathbb{R}$ is monotone if for every $\mathcal{A} \subseteq \mathcal{B} \subseteq \mathcal{N}$,*

$$f(\mathcal{A}) \leqslant f(\mathcal{B}).$$

Note that when we say monotonicity in the context of set functions, we refer to monotone non-decreasing functions by default.

Definition 5. *A set function is said to be normalized if*

$$f(\varnothing) = 0.$$

For monotone non-decreasing and normalized submodular function maximization under cardinality constraints, i.e., for the problem

$$\underset{\mathcal{S}\subseteq\mathcal{N}}{\text{maximize}} f(\mathcal{S}) \quad \text{subject to} \quad |\mathcal{S}| = K, \tag{2.17}$$

the greedy algorithm is an efficient algorithm that provides a near-optimal performance guarantee [32], [28] and thus is widely used. The greedy algorithm is presented in Algorithm 2, and its near optimality is discussed next. Note that in Algorithm 2 and the rest of this section, $\mathcal{S}_k$ denotes the selected set in the kth step.

Algorithm 2 Greedy algorithm for submodular maximization

Input: K, $\mathcal{S}_k = \varnothing$
Output: $\mathcal{S}_K$
1: **for** $k \leftarrow 1$ to K **do**
2: $\quad s^* \leftarrow \arg\max_{s\notin\mathcal{S}_k} f(\mathcal{S}_k \cup \{s\})$
3: $\quad \mathcal{S}_k \leftarrow \mathcal{S} \cup \{s^*\}$
4: **end for**

Theorem 2. *Let $2^{\mathcal{N}} \to \mathbb{R}$ be a normalized monotone submodular function, denote $\mathcal{S}^*$ as the optimal solution of* (2.17)*, and denote the solution obtained by Algorithm 2 as $\mathcal{S}_K$. Then we have*

$$f(\mathcal{S}_K) \geq \left(1 - \frac{1}{e}\right) f(\mathcal{S}^*), \tag{2.18}$$

where e is Euler's number.

Proof. We denote by $\Delta(e \mid S) := f(S\cup\{e\}) - f(S)$ the *discrete derivative* of f at $\mathcal{S}$ with respect to e [28]. Ordering the the elements of $\mathcal{S}^*$ arbitrarily as $\{v_1^*, \ldots, v_K^*\}$, then we have the following inequalities.

$$\begin{aligned}
f(S^*) &\overset{(a)}{\leq} f(S^* \cup S_k) \\
&\overset{(b)}{=} f(\mathcal{S}_k) + \sum_{j=1}^{k} \Delta\left(v_j^* \mid \mathcal{S}_k \cup \left\{v_1^*, \ldots, v_{j-1}^*\right\}\right) \\
&\overset{(c)}{\leq} f(\mathcal{S}_k) + \sum_{v\in S^*} \Delta(v \mid \mathcal{S}_k) \\
&\overset{(d)}{\leq} f(\mathcal{S}_k) + \sum_{v\in S^*} \left(f(S_{i+1}) - f(\mathcal{S}_k)\right) \\
&\overset{(e)}{\leq} f(\mathcal{S}_k) + K\left(f(S_{i+1}) - f(\mathcal{S}_k)\right)
\end{aligned}$$

Here, (a) comes from the monotonicity of f, (b) is the telescoping sum of $f(S^* \cup \mathcal{S}_k)$, (c) holds because of the submodularity of f, (d) is true since $\mathcal{S}_{i+1}$ is selected greedily based on $\mathcal{S}_i$, and (e) obviously holds due to the cardinality constraint $|\mathcal{S}| = K$. Then we have

$$f(\mathcal{S}^*) - f(\mathcal{S}_k) \leq K\left(f(\mathcal{S}_{k+1}) - f(\mathcal{S}_k)\right), \tag{2.19}$$

which can be rearranged as

$$f(\mathcal{S}^*) - f(\mathcal{S}_{k+1})) \leq \left(1 - \frac{1}{K}\right)(f(\mathcal{S}^*) - f(\mathcal{S}_k)), \tag{2.20}$$

and hence $f(\mathcal{S}^*) - f(\mathcal{S}_k \leq \left(1 - \frac{1}{K}\right)^k (f(\mathcal{S}^*) - f(\varnothing)) = \left(1 - \frac{1}{K}\right)^k f(\mathcal{S}^*)$, because f is normalized. Then we have

$$f(\mathcal{S}^*) - f(\mathcal{S}_k) \leq \left(1 - \frac{1}{K}\right)^k f(\mathcal{S}^*) \leq e^{-k/K} f\left(S^*\right). \tag{2.21}$$

The second inequality holds because $1 - x = e^{-x}$. Rearranging the inequality and substituting $k = K$, we obtain the result shown in Theorem 2. □

Besides monotone submodular functions, there is another particular class of submodular functions called symmetric. The symmetry implies that the function values defined on a set and its complement are equal. The definition is given as follows.

Definition 6. *A set function $f : 2^{\mathcal{N}} \to \mathbb{R}$ is called symmetric if*

$$f(\mathcal{S}) = f(\mathcal{N} \setminus \mathcal{S}).$$

3

Graph Node Selection Problem

In this chapter, we go over the problem of sampling the optimal subset on graphs to reconstruct the whole graph signal because the set selection problem is highly relevant to our node partitioning problem. This chapter is organized as this: We first specify the system model of the problem. After this, we give the expression of the optimal interpolation operator for a given sampling subset. Then some ways of scalarization of the error covariance matrix corresponding to the optimal interpolation are discussed. Based on these scalarizations, we formulate the subset selection problem and present two essential techniques to tackle this problem.

3.1. System Model

For the graph signal interpolation in the noisy case, we consider the model

$$\mathbf{y} = \mathbf{x} + \mathbf{w}, \tag{3.1}$$

where $\mathbf{y}, \mathbf{x}, \mathbf{w} \in \mathbb{R}^N$, $\mathbf{x}$ is a K-bandlimited graph signal and $[\mathbf{w}]_1, \ldots, [\mathbf{w}]_N$ are independent identically distributed $\mathcal{N}\left(0, {\sigma_w}^2\right)$ random variables treated as noise. Without loss of generality, we assume that $\sigma_w = 1$, so the noise covariance matrix $\mathbf{\Lambda_w} = \mathbf{I}$. Denote the Laplacian matrix of the graph corresponding to the signal as $\mathbf{L} = \mathbf{U}\mathbf{\Lambda}\mathbf{U}^\top$. We also assume that our graph signal is stationary [33][34][35], i.e., $\mathbf{\Lambda_{\hat{x}}} = \mathbb{E}\hat{\mathbf{x}}_K\hat{\mathbf{x}}_K^\top = \text{diag}\left\{[\boldsymbol{\lambda}_{\hat{\mathbf{x}}}]_1, \ldots [\boldsymbol{\lambda}_{\hat{\mathbf{x}}}]_K\right\}$, where $\mathbf{\Lambda_{\hat{x}}} \in \mathbb{R}^{K\times K}$, $\boldsymbol{\lambda}_{\hat{\mathbf{x}}} \in \mathbb{R}^K$ and $[\mathbf{\Lambda}_{\hat{\mathbf{x}}}]_i > 0, \forall i$.

3.2. Optimal Interpolation Operator

We first derive for a given set $\mathcal{S}$, the corresponding interpolation operator $\mathbf{\Psi}$ of signal $\mathbf{x}$. As defined in chapter 2, denote the sampling operator $\mathbf{\Phi} \in \{0,1\}^{|\mathcal{S}|\times N}$ (the corresponding sampling set is obviously $\mathcal{S}$), so the sampled signal can be written as

$$\mathbf{y}_\mathcal{S} = \mathbf{\Phi}\mathbf{y}, \tag{3.2}$$

and thus the estimated signal $\mathbf{x}'$ is given by

$$\mathbf{x}' = \mathbf{\Psi}\mathbf{y}_\mathcal{S} = \mathbf{\Psi}\mathbf{\Phi}\mathbf{y}, \tag{3.3}$$

for some $\mathbf{\Psi} \in \mathbb{R}^{N\times|\mathcal{S}|}$. So the corresponding interpolation error covariance matrix $\mathbf{K}(\mathbf{x}')$ is thus

$$\mathbf{K}(\mathbf{x}') = \mathbb{E}(\mathbf{x}-\mathbf{x}')(\mathbf{x}-\mathbf{x}')^\top = \mathbb{E}(\mathbf{x}-\mathbf{\Psi}\mathbf{\Phi}\mathbf{y})(\mathbf{x}-\mathbf{\Psi}\mathbf{\Phi}\mathbf{y})^\top. \tag{3.4}$$

So the objective becomes finding the optimal interpolation operator $\mathbf{\Psi}^*$ such that $\mathbf{K}(\mathbf{x}')$ is minimized, i.e., $\mathbb{E}(\mathbf{x}-\mathbf{\Psi}^*\Phi\mathbf{y})(\mathbf{x}-\mathbf{\Psi}^*\Phi\mathbf{y})^\top \preceq \mathbb{E}(\mathbf{x}-\mathbf{\Psi}\mathbf{\Phi}\mathbf{y})(\mathbf{x}-\mathbf{\Psi}\mathbf{\Phi}\mathbf{y})^\top$ for all $\mathbf{\Psi}$, where $\preceq$ denotes smaller or equal to in the matrix inequality. This is a PSD matrix minimization problem that searches for the optimal $\mathbf{\Psi}^*$ that minimizes the error covariance matrix $\mathbf{K}(\mathbf{x}')$. In general, minimizing a PSD cone needs not to have a solution since the ordering of PSD matrices is partial; hence the optimal solution smaller than any other matrix does not necessarily exist [36]. However, [24] shows this is not the case here by finding

the dominant solution $\mathbf{\Psi}^*$. This can be obtained by simultaneously minimizing the scalar cost function below for all b [37], [38].

$$J(\mathbf{\Psi}) = \mathbf{b}^\top \mathbf{K}(\mathbf{x}')\mathbf{b}. \tag{3.5}$$

Comebine (3.3), (3.4), (3.5), we obtain

$$\begin{aligned} J(\mathbf{\Psi}) =& \mathbf{b}^\top \mathbb{E}(\mathbf{x}-\mathbf{x}')(\mathbf{x}-\mathbf{x}')^\top \mathbf{b} \\ =& \mathbf{b}^\top \mathbb{E}\left(\mathbf{U}_K\hat{\mathbf{x}}_K - \mathbf{\Psi}\mathbf{y}_{\mathcal{S}}\right)\left(\mathbf{U}_K\hat{\mathbf{x}}_K - \mathbf{\Psi}\mathbf{y}_{\mathcal{S}}\right)^\top \mathbf{b} \\ =& \mathbf{b}^\top \left[\mathbf{U}_K\mathbf{\Lambda}_{\hat{\mathbf{x}}}\mathbf{U}_K^\top - \mathbf{\Psi}\mathbf{\Phi}\mathbf{U}_K\mathbf{\Lambda}_{\hat{\mathbf{x}}}\mathbf{U}_K^\top - \mathbf{U}_K\mathbf{\Lambda}_{\hat{\mathbf{x}}}\mathbf{U}_K^\top\mathbf{\Phi}^\top\mathbf{\Psi}^\top + \mathbf{\Psi}\mathbf{\Phi}\left(\mathbf{U}_K\mathbf{\Lambda}_{\hat{\mathbf{x}}}\mathbf{U}_K^\top + \mathbf{I}\right)\mathbf{\Phi}^\top\mathbf{\Psi}^\top\right]\mathbf{b}. \end{aligned} \tag{3.6}$$

Set $\frac{\partial J(\mathbf{\Psi})}{\partial \mathbf{b}^\top \mathbf{\Psi}} = \mathbf{0}$, we have

$$\mathbf{\Phi}\left(\mathbf{U}_K\mathbf{\Lambda}_{\hat{\mathbf{x}}}\mathbf{U}_K^\top + \mathbf{\Lambda}_w\right)\mathbf{\Phi}^\top\mathbf{\Psi}^\top\mathbf{b} = \mathbf{\Phi}\mathbf{U}_K\mathbf{\Lambda}_{\hat{\mathbf{x}}}\mathbf{U}_K^\top\mathbf{b}, \tag{3.7}$$

which must hold for all $\mathbf{b}$. Thus, the optimal interpolation operator $\mathbf{\Psi}^*$ satisfies

$$\mathbf{\Psi}^*\mathbf{\Phi}\left(\mathbf{U}_K\mathbf{\Lambda}_{\hat{\mathbf{x}}}\mathbf{U}_K^\top + \mathbf{\Lambda}_{\boldsymbol{w}}\right)\mathbf{\Phi}^\top = \mathbf{U}_K\mathbf{\Lambda}_{\hat{\mathbf{x}}}\mathbf{U}_K^\top\mathbf{\Phi}^\top. \tag{3.8}$$

Therefore,

$$\mathbf{\Psi}^* = \mathbf{U}_K\left(\mathbf{U}_K^\top\mathbf{\Phi}^\top\mathbf{\Phi}\mathbf{\Lambda}_{\boldsymbol{w}}^{-1}\mathbf{\Phi}^\top\mathbf{\Phi}\mathbf{U}_K + \mathbf{\Lambda}_{\hat{\mathbf{x}}}^{-1}\right)^{-1}\mathbf{U}_k^\top\mathbf{\Phi}^\top. \tag{3.9}$$

Then plug $\mathbf{\Psi}^*$ into (3.4) and use matrix inversion lemma [19] we have

$$\mathbf{K}(\mathbf{x}'^*) = \mathbf{U}_K\left(\mathbf{\Lambda}_{\hat{\mathbf{x}}}^{-1} + \mathbf{U}_K^\top\mathbf{\Phi}^\top\mathbf{\Phi}\mathbf{\Lambda}_{\boldsymbol{w}}^{-1}\mathbf{\Phi}^\top\mathbf{\Phi}\mathbf{U}_K\right)^{-1}\mathbf{U}_K^\top. \tag{3.10}$$

Indeed, now the interpolation error covariance matrix can be expressed in terms of the sampling operator $\mathbf{\Phi}$ solely. In other words, the error covariance matrix only depends on the sampling set chosen in our setting since the interpolation operator is fixed for a given subset.

3.3. Scalarizations

This section introduces some scalarizations that allow us to evaluate the error covariance matrix obtained above. Denote $\mathbf{K}(\mathbf{x}'^*)$ as $\mathbf{K}^*$ for simplicity. To qualify how good the subset $\mathcal{S}$ that is selected, several scalarizations could be used [36], and we are interested in minimizing these scalarizations.

A-optimal design A-optimal design can also be referred to as mean square error (MSE):

$$\mathrm{tr}\{\mathbf{K}^*\}. \tag{3.11}$$

D-optimal design D-optimal design is the most widely used scalarization, which can be interpreted as the log volume of the confidence ellipsoid [36]. It is defined as

$$\log\det\{\mathbf{K}^*\}. \tag{3.12}$$

E-optimal design E-optimal can also be called the worst-case error variance [39], which refers to the norm of the error covariance matrix, i.e., the maximum eigenvalue of $\mathbf{K}^*$.

$$\lambda_{\max}\{\mathbf{K}^*\}. \tag{3.13}$$

If we look at the D-optimal design, the rank-deficiency issue exists for $\mathbf{K}^*$ is always singular regardless of which subset is selected. Because of this, the determinant of the error covariance matrix of $\mathbf{x}'$ is always zero.

To deal with the rank-deficiency issue, we show in Proposition 1 that for a given observed subset, reconstructing $\hat{\mathbf{x}}_K$ using the optimal interpolation operator, and then interpolating $\mathbf{x}$ as $\mathbf{x}' = \mathbf{U}_K\hat{\mathbf{x}}'^*_K$ is equivalent to interpolating $\mathbf{x}$ directly using $\mathbf{\Psi}^*$, where $\hat{\mathbf{x}}'^*_K$ denotes the optimal estimator of $\hat{\mathbf{x}}_K$. By the proposition, finding the optimal subset for reconstructing $\mathbf{x}$ is equivalent to finding the optimal subset for reconstructing $\hat{\mathbf{x}}_K$, and because the error covariance matrix for $\hat{\mathbf{x}}'_K$ is always full rank, we can thus overcome the rank-deficiency problem by working on the error covariance matrix for $\hat{\mathbf{x}}'_K$ instead of $\mathbf{x}'$.

Proposition 1. *Let $\mathbf{\Psi}_0$ denote the optimal interpolation operator (applied on $\mathbf{y}_{\mathcal{S}}$) for reconstructing $\hat{\mathbf{x}}_K$, $\mathbf{\Psi}^*$ denote the optimal interpolation operator for reconstructing* $\mathbf{x}$ *as defined in* (3.8)*, then $\mathbf{\Psi}^* = \mathbf{U}_K\mathbf{\Psi}_0$.*

Proof. Again, consider the model defined in (3.1) $\mathbf{y} = \mathbf{x} + \mathbf{w} = \mathbf{U}_K\hat{\mathbf{x}}_K + \mathbf{w}$, which can also be written as

$$[\mathbf{y}]_i = \tilde{\tilde{\mathbf{u}}}_i^\top \hat{\mathbf{x}}_K + [\mathbf{w}]_i, \quad i = 1, \dots, N, \tag{3.14}$$

where $\mathbf{U}_K = [\tilde{\mathbf{u}}_1 \cdots \tilde{\mathbf{u}}_N]^\top$, and the $\mathbf{w}$ is Gaussian white noise. Thus, the optimal estimator of $\hat{\mathbf{x}}_K$ is given by [40]

$$\hat{\mathbf{x}}_K'^* = \left(\sum_{i\in\mathcal{S}} \tilde{\mathbf{u}}_i\tilde{\mathbf{u}}_i^\top + \mathbf{\Lambda}_{\hat{\mathbf{x}}}^{-1}\right)^{-1} \sum_{i\in\mathcal{S}} [\mathbf{y}]_i\tilde{\mathbf{u}}_i, \tag{3.15}$$

or equivalently,

$$\hat{\mathbf{x}}_K'^* = \left(\mathbf{U}_K^\top\mathbf{\Phi}^\top\mathbf{\Phi}\mathbf{\Lambda}_{\boldsymbol{w}}^{-1}\mathbf{\Phi}^\top\mathbf{\Phi}\mathbf{U}_K + \mathbf{\Lambda}_{\hat{\mathbf{x}}}^{-1}\right)^{-1} \mathbf{U}_k^\top\mathbf{\Phi}^\top\mathbf{\Phi}\mathbf{y}. \tag{3.16}$$

Hence, the corresponding interpolation operator $\mathbf{\Psi}_0$ for reconstructing $\hat{\mathbf{x}}_K$ is given by

$$\mathbf{\Psi}_0 = \left(\mathbf{U}_K^\top\mathbf{\Phi}^\top\mathbf{\Phi}\mathbf{\Lambda}_{\boldsymbol{w}}^{-1}\mathbf{\Phi}^\top\mathbf{\Phi}\mathbf{U}_K + \mathbf{\Lambda}_{\hat{\mathbf{x}}}^{-1}\right)^{-1} \mathbf{U}_k^\top\mathbf{\Phi}^\top. \tag{3.17}$$

Combine (3.9) and (3.17), we obtain $\mathbf{\Psi}^* = \mathbf{U}_K\mathbf{\Psi}_0$, which completes the proof. □

According to (3.15), the error covariance matrix of $\hat{\mathbf{x}}_K'^*$ is then given by

$$\mathbb{E}(\hat{\mathbf{x}}_K'^* - \hat{\mathbf{x}}_K)(\hat{\mathbf{x}}_K'^* - \hat{\mathbf{x}}_K)^\top = \left(\sum_{i\in\mathcal{S}} \tilde{\mathbf{u}}_i\tilde{\mathbf{u}}_i^\top + \mathbf{\Lambda}_{\hat{\mathbf{x}}}^{-1}\right)^{-1}, \tag{3.18}$$

which is full rank for all $\mathcal{S}$. Thus in the remainder of this report, it is more convenient to scalarize the error covariance matrix of $\hat{\mathbf{x}}_K'$ because the corresponding D-optimal design is well-defined then (A-optimal and E-optimal are well-defined as well obviously).

3.4. Problem formulation for set selection on graphs

In the previous section, we show three optimal designs which can be used to evaluate the optimal interpolator $\mathbf{\Psi}^*$ that minimizes the estimation error covariance matrix for a given sampling subset. However, it is not guaranteed that there is no other sampling subset of the same size for which the interpolation quality is better. To handle this problem, we investigate the set selection problem to find the optimal subset for a given size. Formally, we formulate the sampling set selection problem in the following manner.

$$\begin{array}{ll} \text{minimize} & g(\mathcal{S}) \\ \text{subject to} & |\mathcal{S}| = k. \end{array} \tag{3.19}$$

We can vary function g according to the type of optimal design. Based on A-optimal design: $g(\mathcal{S}) = \mathrm{tr}\{\left(\sum_{i\in\mathcal{S}} \tilde{\mathbf{u}}_i\tilde{\mathbf{u}}_i^\top + \mathbf{\Lambda}_{\hat{\mathbf{x}}}^{-1}\right)^{-1}\}$. Based on D-optimal design: $g(\mathcal{S}) = \log\det\{\left(\sum_{i\in\mathcal{S}} \tilde{\mathbf{u}}_i\tilde{\mathbf{u}}_i^\top + \mathbf{\Lambda}_{\hat{\mathbf{x}}}^{-1}\right)^{-1}\}$. Based on E-optimal design: $g(\mathcal{S}) = \lambda_{\max}\{\left(\sum_{i\in\mathcal{S}} \tilde{\mathbf{u}}_i\tilde{\mathbf{u}}_i^\top + \mathbf{\Lambda}_{\hat{\mathbf{x}}}^{-1}\right)^{-1}\}$.

3.5. Algorithms

(3.20) is a combinatorial problem where obtaining the exact solution is NP-hard. In this section, we review the two most commonly used methods, the greedy and the convex approaches, to find an approximate solution for this problem. We take D-optimal design as an example for simplicity, but extending these approaches to A-optimal and E-optimal is easy.

3.5.1. Greedy Algorithm

For D-optimal design, we can rewrite the problem as

$$\begin{array}{ll} \text{maximize} & f(\mathcal{S}) = \log\det\{\mathbf{\Lambda}_{\hat{\mathbf{x}}}^{-1} + \sum_{i\in\mathcal{S}} \tilde{\mathbf{u}}_i\tilde{\mathbf{u}}_i^\top\} - \log\det\{\mathbf{\Lambda}_{\hat{\mathbf{x}}}^{-1}\} \\ \text{subject to} & |\mathcal{S}| = k, \end{array} \tag{3.20}$$

where $\log\det\{\mathbf{\Lambda}_{\hat{\mathbf{x}}}^{-1}\}$ is a normalization constant such that $f(\varnothing) = 0$.

The greedy set selection algorithm is presented in Algorithm 3, a straightforward application of Algorithm 2. In [41], the authors show that $f(\mathcal{S})$ is a submodular monotone function, and we present their result in the following lemma. Therefore, by Theorem 2, a $1 - \frac{1}{e}$ performance lower bound is guaranteed by applying Algorithm 3.

Algorithm 3 Greedy algorithm for sensor selection

Input: k, $\mathcal{S}_1 = \varnothing$
Output: $\mathcal{S}_k$
1: **for** $i \leftarrow 1$ to k **do**
2: $\quad s^* \leftarrow \arg\max_{s\notin\mathcal{S}_i} \log\det\{\mathbf{\Lambda}_{\hat{\mathbf{x}}}^{-1} + \sum_{j\in\mathcal{S}_i} \tilde{\mathbf{u}}_j\tilde{\mathbf{u}}_j^\top + \tilde{\mathbf{u}}_s\tilde{\mathbf{u}}_s^\top\}$
3: $\quad \mathcal{S}_i \leftarrow \mathcal{S}_i \cup \{s^*\}$
4: **end for**

Lemma 1. *$f(\mathcal{S}) = \log\det\{\mathbf{\Lambda}_{\hat{\mathbf{x}}}^{-1} + \sum_{i\in\mathcal{S}} \tilde{\mathbf{u}}_i\tilde{\mathbf{u}}_i^\top\}$ is a monotone submodular function.*

Proof. See [41]. □

3.5.2. Convex Optimization

The second approach is based on thresholding the solution of the convex relaxation of the original problem. Though no performance guarantee is yet found for this approach, numerical results in [25] show that the gap between the approximated and optimal solutions is small. To formulate the corresponding convex relaxation more efficiently, we first rewrite the original problem as

$$\begin{array}{ll} \text{maximize} & \log\det\{\sum_{i=1}^N z_i\tilde{\mathbf{u}}_i\tilde{\mathbf{u}}_i^\top + \mathbf{\Lambda}_{\hat{\mathbf{x}}}^{-1}\} - \log\det\{\mathbf{\Lambda}_{\hat{\mathbf{x}}}^{-1}\} \\ \text{subject to} & z_i \in \{0,1\}, \sum_{i=1}^m z_i = k. \end{array} \tag{3.21}$$

The relaxed convex problem is thus given by

$$\begin{array}{ll} \text{maximize} & \log\det\{\sum_{i=1}^N z_i\tilde{\mathbf{u}}_i\tilde{\mathbf{u}}_i^\top + \mathbf{\Lambda}_{\hat{\mathbf{x}}}^{-1}\} - \log\det\{\mathbf{\Lambda}_{\hat{\mathbf{x}}}^{-1}\} \\ \text{subject to} & 0 \le z_i \le 1, \quad i = 1,2,\ldots,N \\ & \sum_{i=1}^m z_i = k, \end{array} \tag{3.22}$$

which can be solved by interior point method efficiently [25], [36]. Once we obtain the solution of (3.22) (which is fractional in general), we threshold on the fractional solution to get the binary solution back.

4

Problem Formulation and Related Work

In this chapter, based on the node selection problem, we first formulate the node partitioning problem. Then we review some other problem formulations for the node partitioning problem and show some existing algorithms that aim to solve these problems. We will discuss these algorithms in depth and explain why they are inferior. After this, we compare our problem formulation with the problem formulations in these literature and provide reasons why ours is more appropriate because near-optimal solutions can be found due to the structure of our cost function.

4.1. Problem Formulation

In this section, we formally formulate the node partitioning problem. Our cost function is given by

$$\begin{array}{ll} \text{maximize} & f\left(\mathcal{S}_1\right)+\cdots+f\left(\mathcal{S}_P\right) \\ \text{subject to} & \mathcal{S}_1 \cup \cdots \cup \mathcal{S}_P=\mathcal{N} \\ & \mathcal{S}_i \cap \mathcal{S}_j=\varnothing, \forall i, j, \end{array} \tag{4.1}$$

where $f(\mathcal{S}_p) = \log\det\{\mathbf{\Lambda}_{\tilde{\mathbf{x}}}^{-1} + \sum_{i\in\mathcal{S}_p} \tilde{\mathbf{u}}_i\tilde{\mathbf{u}}_i^\top\} - \log\det\{\mathbf{\Lambda}_{\tilde{\mathbf{x}}}^{-1}\}, p = 1, \cdots, P$, $\mathcal{N}$ is the ground set of the node, and $\log\det\{\mathbf{\Lambda}_{\tilde{\mathbf{x}}}^{-1}\}$ is a normalized constant that guarantees that each part of the summation is always non-negative. In [42], the term *representative sampling subsets* refers to these disjoint subsets indicating that they are representatives for reconstructing the entire graph signal.

4.2. Related Work

As discussed in Chapter 3, for the graph node sampling problem, two typical approximated approaches are widely used. The first is the greedy approach, where we select sensors sequentially and always pursue the locally optimal one. The overall performance can be nicely guaranteed with a factor $(1 - \frac{1}{e})$ when D-optimal is used. Several literature are relating to this topic [43], [24], [41]. The second heuristic is based on convex optimization, where binary constraints are relaxed to be convex, then threshold the convex solution to give back the binary solution [25]. In this work, instead of finding one subset, we aim to find a partition of the graph such that each disjoint subset is sufficiently good for interpolation. Likewise, existing literature also attempts to tackle the problem following these two types of heuristics: [44], [42] proposed greedy algorithms to solve the partitioning problem while [39] used a convex optimization based method. We next show several partition algorithms proposed in the above literature and have some comments about why they are not good enough, especially the convex approach, which does not work. We remark here that though many of the algorithms use A-optimal(MSE) design as the performance measure, they can be easily extended to D-optimal design by replacing the $\mathrm{tr}\{\cdot\}$ with $\log\det\{\cdot\}$, which has no impact on the algorithm implementation. Hence these algorithms can be adapted to compare our proposed algorithms, as we will show later.

4.2.1. Problem Formulation Based on Optimizing the Worst Subset

In [39] and [42], the node partitioning problem is formulated as this

$$\begin{aligned} &\text{minimize } t \\ &\text{subject to } \mathrm{tr}\{(\sum_{i\in\mathcal{S}_p} \tilde{\mathbf{u}}_i\tilde{\mathbf{u}}_i^\top + \mathbf{\Lambda}_{\hat{\mathbf{x}}}^{-1})^{-1}\} \leq t \quad \forall p = 1,\ldots,P, \end{aligned} \tag{4.2}$$

i.e., minimizing the reconstruction MSE corresponding to the worst subset.

To the best of our knowledge, for problem (4.2), [39] relax the binary constraint of the original problem and try to obtain a sparse solution via *iterative reweighted* l_1 technique; [42] propose a greedy heuristic. We next discuss these two approaches in detail.

Iterative Reweighted l_1 algorithm

We start with the convex-based approach, which does not work, as we will show later. For convenience, let us define a partitioning matrix $\mathbf{Z}$

$$\mathbf{Z} = \begin{bmatrix} \mathbf{z}_1 & \mathbf{z}_2 & \ldots & \mathbf{z}_P \end{bmatrix} \in \{0,1\}^{N\times P}, \tag{4.3}$$

where entries in each column specify which nodes are sampled in each group. P columns implies that we have P partitions in total. Since each node must be activated exactly once, $\mathbf{Z}$ is a binary matrix with a single "one" entry in each row, and the other entries are all zeros. Obviously, $\sum_{i\in\mathcal{S}_p} \tilde{\mathbf{u}}_i\tilde{\mathbf{u}}_i^\top = \mathbf{U}_K^\top \mathrm{diag}\{\mathbf{z}_p\}\mathbf{U}_K, \forall p \in \{1,\cdots,P\}$, so we obtain the equivalent optimization problem as (4.2)

$$\begin{aligned} &\underset{y,\mathbf{Z}\in\{0,1\}^{N\times P}}{\text{minimize}} \; y \\ &\text{subject to } \sum_{p=1}^{P} \mathbf{z}_p = \mathbf{1} \\ &\qquad \mathrm{tr}\left\{\left(\mathbf{U}_K^\top \mathrm{diag}\left\{\mathbf{z}_p\right\}\mathbf{U}_K + \mathbf{\Lambda}_{\hat{\mathbf{x}}}^{-1}\right)^{-1}\right\} \leqslant y, \forall p = 1,\ldots,P. \end{aligned} \tag{4.4}$$

The first constraint in (4.4) guarantees that each node is selected precisely once. The second constraint introduces a slack variable, y, that helps to optimize the cost function. This problem is an NP-hard mixed-integer semidefinite optimization problem and is hard to solve.

To make the problem convex, we relax the binary constraint.

$$\begin{aligned} &\underset{y,\mathbf{Z}\in[0,1]^{N\times P}}{\text{minimize}} \; y \\ &\text{subject to } \sum_{p=1}^{P} \mathbf{z}_p = \mathbf{1} \\ &\qquad \mathrm{tr}\left\{\left(\mathbf{U}_K^\top \mathrm{diag}\left\{\mathbf{z}_p\right\}\mathbf{U}_K + \mathbf{\Lambda}_{\hat{\mathbf{x}}}^{-1}\right)^{-1}\right\} \leqslant y, \forall p = 1,\ldots,P. \end{aligned} \tag{4.5}$$

The solution of (4.5) is given by $y^* = \mathrm{tr}\{(\frac{1}{P}\mathbf{I} + \mathbf{\Lambda}_{\boldsymbol{x}}^{-\mathbf{1}})^{-1}\}$, and $[\mathbf{Z}^*]_{i,j} = \frac{1}{P}, \forall i,j = 1,\cdots,N$. Because all the entries are the same, techniques such as thresholding fail to work here.

In [39], the authors add a weighted l_1 penalty term the promote sparse solutions

$$\begin{aligned} &\underset{y,\mathbf{Z}\in[0,1]^{N\times P}}{\text{minimize}} \; y + \lambda\sum_{p=1}^{P} \mathbf{w}_p^\top \mathbf{z}_p \\ &\text{subject to } \sum_{p=1}^{P} \mathbf{z}_p = \mathbf{1} \\ &\qquad \mathrm{tr}\left\{\left(\mathbf{U}_K^\top \mathrm{diag}\left\{\mathbf{z}_p\right\}\mathbf{U}_K + \mathbf{\Lambda}_{\hat{\mathbf{x}}}^{-1}\right)^{-1}\right\} \leqslant y, \forall p = 1,\ldots,P, \end{aligned} \tag{4.6}$$

and to further promote sparsity hopefully, a reweighted l_1 minimization method presented in [45] is used. Here we briefly illustrate the idea behind this sparsity-enhancing algorithm.

Consider an l_0 minimization problem

$$\min_{\mathbf{x}\in\mathbb{R}^n} \|x\|_0 \text{ subject to } \quad \mathbf{y} = \mathbf{\Phi x}, \tag{4.7}$$

which is non-convex and extremely hard to solve. A common heuristic for such a problem is to replace the l_0 norm with an l_1 norm so the problem becomes

$$\min_{\mathbf{x}\in\mathbb{R}^n} \|x\|_1 \text{ subject to } \quad \mathbf{y} = \mathbf{\Phi x}, \tag{4.8}$$

which also promotes sparsity of **x** [46]. However, these two problems are not equivalent in general. A key difference is a dependence on magnitude [45]: larger entries are penalized more heavily than smaller entries in the l_1 norm. In contrast, in the l_0 norm, all nonzero entries are penalized equally. The solution of this issue could be solved by the following weighted l_1 norm minimization problem

$$\min_{\mathbf{x}\in\mathbf{R}^n} \sum_{i=1} w_i \left|x_i\right| \quad \text{ subject to } \quad \mathbf{y} = \mathbf{\Phi x}, \tag{4.9}$$

if the weights are set properly. It is for sure that if we set the weights to be inversely proportional to the corresponding entries of x, then the solution should exactly be the same as that of l_0. But without any knowledge of x, it is impossible to define such weights. The authors in [45] thus propose an iterative algorithm that alternates between estimating **x** and redefining the weights. The algorithm is given by

Algorithm 4 Iterative reweighted l_1 approach

Input: **y**, Φ, maximum number of iterations l_{max}, stability parameter ϵ
Output: **x**

1: Set $l = 0$ and $w_i^{(0)} = 1, i = 1, \ldots, n$
2: **while** $l < l_{max}$ or **x** has not been converged **do**
3: Solve the minimization problem

$$\mathbf{x}^{(\ell)} = \arg\min \left\|\mathbf{W}^{(\ell)}\mathbf{x}\right\|_1 \quad \text{ subject to } \quad \mathbf{y} = \mathbf{\Phi x},$$

where $\mathbf{W}^{(\ell)} = diag(\mathbf{w}^{(\ell)})$.
4: Update the weights

$$w_i^{(\ell+1)} = \frac{1}{\left|x_i^{(\ell)}\right| + \epsilon}$$

5: **end while**

To apply algorithm 4 to solve the node partitioning problem, the procedure is similar, we first initialize the weighted matrix $\mathbf{W} = [\mathbf{w}_1, \cdots, \mathbf{w}_P]$ given in (4.6) with some preset weights, then solve the convex optimization problem (4.6). Once the solution is obtained, update the weights in the style shown in algorithm 4 and solve the optimization problem again. We then repeat this procedure until some stopping criterion is satisfied. This scheme is shown in algorithm 5.

In [39], they initialize the weights matrix **W** with all-one entries. However, this initialization will never promote the sparsity of the solution. To be more general, any all-constant initialization does not promote sparsity, and here is the reason. Without loss of generality, we now assume that $[\mathbf{W}]_{i,j} = 1, \forall i, j$. According to the first constraint of the problem (4.6),

$$\sum_{p=1}^{P} \mathbf{z}_p = \mathbf{1}, \tag{4.10}$$

which implies that every node is active only once. However, there is an implicit constraint

$$\sum_{\forall i,j} [\mathbf{Z}]_{ij} = N, \tag{4.11}$$

Algorithm 5 Iterative reweighted l_1 approach based node partitioning

Input: total number of nodes m, number of partitions P, regularization parameter $\lambda > 0$, $\mathbf{U}_K$, ϵ, β
Output: indicator matrix $\mathbf{Z} \in \{0,1\}^{N \times P}$

```
Initialize the weights of W with some values and denote as W^(0)
Solve (4.6) with the weight matrix W^(0), denote the solution as Z^(0).
Updating the weights of W^(0) in the style of algorithm 4 and denote the updated one as W^(1)
Solve (4.6) with the weight matrix W^(1), denote the solution as Z^(1).
let l = 1
while Z^(l) − Z^(l−1) ⩾ β do
    Updating the weights of W^(l) in the style of algorithm 4 and denote the updated one as W^(l+1)
    Solve (4.6) with the weight matrix W^(l+1), denote the solution as Z^(l+1)
    l ← l + 1
end while
Threholding to make Z matrix binary.
```

which can be directly derived from (4.10). It is obvious that (4.11) is equivalent to

$$\sum_{p=1}^{P} \mathbf{1}^\top \mathbf{z}_p = N, \tag{4.12}$$

which implies that if the weights are set to be constant, the l_1 penalty term of (4.6) will always be a constant as well, hence it is redundant and fail to promote sparsity (as we know from the previous result, the solution is given by $[\mathbf{Z}^*]_{i,j} = \frac{1}{P}, \forall i,j = 1, \cdots, N$. On the other hand, if a non-constant valued weighted matrix $\mathbf{W}$ is used as the initialization, Algorithm 5 can end up with a sparse solution. However, doing so means that for each subset, we have some preference for each node. If the weights are badly set, then the solution is probably bad as well; If the weights are properly set, then we must be cheating since nobody knows whether an initialization of $\mathbf{W}$ matrix is good or bad without knowing the optimal solution of the original problem (4.4). Hence, this approach is not a good choice for (4.2).

Simultaneous Iterative Partitioning

In Simultaneous Iterative Partitioning (SIP), the idea is to generate subsets such that the MSE of each subset is balanced. Given the number of subsets P, at each iteration, SIP creates the representative sampling subsets simultaneously. It works as this, let $(1, 2, \cdots, N)$ denote the set of node $\mathcal{N}$ as defined; at the first step, we assort the nodes based on

$$\text{MSE}\,(s) = \text{tr}\left\{\left(\mathbf{\Lambda}_{\tilde{\mathbf{x}}}^{-1} + \tilde{\mathbf{u}}_s \tilde{\mathbf{u}}_s^\top\right)^{-1}\right\}, \forall s \in \{1, \cdots, N\}. \tag{4.13}$$

The P nodes with the smallest MSE are first assigned the P representative sampling subsets, one for each. Then at each iteration, for the subset with the largest MSE, say $\mathcal{S}_j$, we add the best node m to this set based on

$$v^* = \arg\min\ \text{MSE}\,(\mathcal{S}_j \cup s), \forall s \in \{1, \cdots, N\}, \tag{4.14}$$

and repeat the process until every node is assigned to one subset. The algorithm is shown below.

Now we discuss the problems of this algorithm. Combined with the results we got in the previous section, this algorithm makes sense to some extent, as it promotes the function values $f(\mathcal{S}_p), \forall p$ to be close. However, the solution obtained by this algorithm does not always perform well. Here is a toy example to explain this. Assume that the signal is homoscedastic, i.e.,$\mathbf{\Lambda}_{\tilde{\mathbf{x}}} = \sigma_x \mathbf{I}$, the number of subsets P is set to 2, and the bandwidth $K = 2$. Hence, $\sum_{i \in \mathcal{N}} \tilde{\mathbf{u}}_i \tilde{\mathbf{u}}_i^\top = \mathbf{I} \in \mathbb{R}^{2 \times 2}$. Under this setting, assume we have two different partitions; the first is $\{\mathcal{X}_1, \mathcal{X}_2\}$, and the other is $\{\mathcal{Y}_1, \mathcal{Y}_2\}$, and indeed, $\mathcal{X}_2 = \mathcal{N} \setminus \mathcal{X}_1$, $\mathcal{Y}_2 = \mathcal{N} \setminus \mathcal{Y}_1$. For the first partition, we have $\sum_{i \in \mathcal{X}_1} \tilde{\mathbf{u}}_i \tilde{\mathbf{u}}_i^\top = \sum_{i \in \mathcal{X}_2} \tilde{\mathbf{u}}_i \tilde{\mathbf{u}}_i^\top = \begin{bmatrix} \frac{1}{2} & 0 \\ 0 & \frac{1}{2} \end{bmatrix}$; for the second partition, $\sum_{i \in \mathcal{Y}_1} \tilde{\mathbf{u}}_i \tilde{\mathbf{u}}_i^\top = \begin{bmatrix} 0.9 & 0 \\ 0 & 0.1 \end{bmatrix}$, $\sum_{i \in \mathcal{Y}_2} \tilde{\mathbf{u}}_i \tilde{\mathbf{u}}_i^\top = \begin{bmatrix} 0.1 & 0 \\ 0 & 0.9 \end{bmatrix}$. Obviously, in both cases, the cost function values for each subset are equal, i.e., $f(\mathcal{X}_1) = f(\mathcal{X}_2)$ and $f(\mathcal{Y}_1) = f(\mathcal{Y}_2)$. However, it is for sure that $f(\mathcal{X}_1) + f(\mathcal{X}_2) \geqslant f(\mathcal{Y}_1) + f(\mathcal{Y}_2)$ from the aforementioned results. However,

Algorithm 6 Simultaneous Iterative Partitioning (SIP)

Input: $P, \mathbf{\Lambda}_{\boldsymbol{w}}, \mathbf{\Lambda}_{\hat{\mathbf{x}}}, \mathbf{U}, \mathbf{U}_K$
Output: $\{\mathcal{S}_1, \cdots, \mathcal{S}_P\}$ the complete Partition
1: list of available nodes $\mathcal{L}_{index} = \{1, \cdots, N\}$
2: generate $\mathcal{L}$, where $\mathcal{L}(s) = \text{MSE}(s), \forall s \in \mathcal{L}_{index}$
3: **for** $i \leftarrow 1$ to P **do**
4: $m = \arg\min(\mathcal{L})$
5: $\mathcal{S}_i \leftarrow \varnothing \cup m$
6: remove m from the list $\mathcal{L}_{index}$
7: **end for**
8: **while** $\mathcal{L}_{index} \neq \varnothing$ **do**
9: $j = \arg\max \text{MSE}(\mathcal{S}_j)$
10: $m = \arg\min \text{MSE}(\mathcal{S}_j \cup s), \forall s \in \mathcal{L}_{index}$
11: $\mathcal{S}_j \leftarrow \mathcal{S}_j \cup m$
12: remove m from the list $\mathcal{L}_{index}$
13: **end while**

by SIP algorithm, we cannot make sure which types of solution it would return as it only focuses on the cost value while does not consider how $\sum_{i \in \mathcal{S}} \tilde{\mathbf{u}}_i \tilde{\mathbf{u}}_i^\top$ behaves for a subset $\mathcal{S}$. Besides, SIP has no performance guarantee.

4.2.2. Problem Formulation Based on Maximizing the number of qualified subsets

In [44], the node partitioning problem is formulated as

$$\begin{aligned} &\text{maximize } P \\ &\text{subject to } \text{tr}\{(\sum_{i \in \mathcal{S}_p} \tilde{\mathbf{u}}_i \tilde{\mathbf{u}}_i^\top + \mathbf{\Lambda}_{\hat{\mathbf{x}}}^{-1})^{-1}\} \leq \epsilon \quad \forall p = 1, \ldots, P, \end{aligned} \tag{4.15}$$

that is, given an MSE bound that every subset has to satisfy, we maximize the number of subsets.

Joint Iterative Partitioning

The algorithm proposed in [44] is called *Joint Iterative Partitioning* Algorithm provided in [44]. The idea behind this approach is that we select nodes for a subset greedily until its MSE satisfies the bound we set, then we move on to the next subset and repeat this procedure until every node is assigned. If MSE corresponding to the nodes for the last subset is larger than ϵ, we split these nodes into the previous groups. The pseudo-code is shown in Algorithm 7. The notations are defined as follows: $\tilde{\mathbf{u}}_i$ denotes the transpose of the ith row of $\mathbf{U}_K$, $\mathcal{S}_{p,j}$ denotes the node set of the pth subset in the jth iteration, and $pop(\mathcal{L}\_index_{-1})$ denotes the operation that removes the last item in the list $\mathcal{L}$ and outputs that item. As defined, MSE represents the function where the input is a set, and the output is the interpolation mean square error corresponding to the input set.

Though this algorithm can give us a partition of the nodes, problems exist. Although this algorithm is called Joint Iterative Partitioning, it partitions the nodes sequentially, as the best nodes are first selected for the first subset. The best remaining nodes are selected for the second subset, and so on. Each subset's size in this pattern will become more extensive since the remaining nodes are not as good as the selected one. Therefore, one phenomenon that is very likely to occur is that many "bad" nodes are left for the last subset. They cannot satisfy the preset MSE bound, so we have to split them into the previous subset, which results in a tremendous waste of nodes. In addition, JIP has no performance guarantee.

4.3. Benefits of Our Problem Formulation

Here we highlight some advantages of formulating the problem as (4.1) compared to the above two problem formulations.

- We choose D-optimal as a subset's performance measure because it is submodular in that set. Hence properties of submodular functions can be leveraged in algorithm design.

Algorithm 7 Joint Iterative Partitioning (JIP)

Input: $\mathbf{\Lambda}_{\boldsymbol{w}}, \mathbf{\Lambda}_{\hat{\mathbf{x}}}, \mathbf{U}, \mathbf{U}_K, \epsilon$
Output: $\{\mathcal{S}_1, \cdots, \mathcal{S}_P\}$ the complete Partition
1: $K_0 = \mathbf{\Lambda}_{\hat{\mathbf{x}}}$, $p = 1$, $j = 1$
2: $\mathcal{L}$, where $L(s) = \text{MSE}\,(s)$, $\forall s = 1, \ldots, N$
3: $\mathcal{L}_{index} = \text{argsort}(\mathcal{L})$ (smallest first)
4: **while** $\mathcal{L}_{index} \neq \varnothing$ **do**
5: $v = pop(\mathcal{L}_{index}(1))$
6: $\mathcal{S}_{p,j} = v$
7: **while** $\text{MSE}\,(\mathcal{S}_{p,j}) > \epsilon$ and $\mathcal{L}_{index} \neq \varnothing$ **do**
8: **for** i in $\mathcal{L}_{index}$ **do**
9: **if** $\text{MSE}\,(\mathcal{S}_{p,j} \cup v_i) < \epsilon$ **then**
10: $\mathcal{S}_{p,j+1} = \{\mathcal{S}_{p,j} \cup i\}$
11: $j = 0, p = p + 1$, delete $(\mathcal{L}_{-}\,\text{index} = i)$
12: goto 4
13: **end if**
14: **end for**
15: $\mathcal{S}_{p,j+1} = \{\mathcal{S}_{p,j} \cup \arg\min(\text{MSE}(i \cup \mathcal{S}_{p,j}))\}, \forall i \in \mathcal{L}_{index}$
16: remove chosen node from $\mathcal{L}_{index}$, $j = j + 1$
17: **end while**
18: **end while**
19: **if** $\text{MSE}\,(\mathcal{S}_{p,j} \cup v_i) > \epsilon$ **then**
20: Split the remaining nodes among the other sets uniformly
21: **end if**

- Instead of optimizing the worst subset as (4.2), we optimize the summation of the cost function for each subset. Using this cost function simplifies the constraint since a slack variable is no longer needed.
- We fix P instead of fix a bound as shown in (4.15). This is because for problem (4.15), the optimal solution may not be unique. For example, for some certain bound, the maximum number of subsets $P = 5$, there might be multiple partitions with size five that satisfy the criterion, and all of them are the optimal solutions. However, if we fix P and maximize the cost function, the solution is unique in general.

5

Partitioning Algorithms

In Chapter 4, we formulate the partitioning problem as maximizing the sum of several normalized submodular monotone functions. Such formulation turns the partitioning problem into a *combinatorial auctions*, or *Submodular Welfare* (SW) problem where we have N items and P players with some utility functions, and the utility functions are assumed to be monotone and submodular. The goal is to find an allocation of disjoint sets of items to maximize the sum of utility functions for all players [47], [48], [49], [50]. In our case, the problem (4.1) is a particular SW problem where the "items" are the nodes, the "players" are the representative subsets, and the utility functions f are the same for all subsets. Hence, we can extend algorithms for SW problems to our partitioning problem. In addition, we propose a new greedy algorithm for approximating the problem (4.1).

Note that in the following of the report, we assume that $f(\mathcal{S}) = \log\det\{\mathbf{\Lambda}_{\hat{\mathbf{x}}}^{-1} + \sum_{i\in\mathcal{S}} \tilde{\mathbf{u}}_i\tilde{\mathbf{u}}_i^\top\} - \log\det\{\mathbf{\Lambda}_{\hat{\mathbf{x}}}^{-1}\}$ by default. We call $f(\mathbf{S})$ the utility function for subset $\mathcal{S}$ to avoid confusion with the cost function, which is the sum of all the utility functions.

5.1. Deterministic greedy approaches

We introduce three deterministic greedy algorithms to tackle the partitioning problem (4.1). Algorithm 8 aims to tackle the problem from the perspective of maximizing a submodular function in the special case where $P = 2$. It yields a $\frac{1}{2}$ approximation ratio. Algorithm 9 is an extension of Algorithm 8 where the bi-partitioning procedure is recursively implemented to achieve multi-way partitioning. Though no performance guarantee can be found, numerical results show that this algorithm works well. Algorithm 10 aims to find the disjoint subsets jointly, which gives an approximation ratio of $\frac{1}{2}$ for any P.

5.1.1. Two-Sided greedy algorithm

We first consider the particular case when $P = 2$. In this case, the problem becomes

$$\text{maximize} \quad \log\det\{\mathbf{\Lambda}_{\hat{\mathbf{x}}}^{-1} + \sum_{i\in\mathcal{S}} \tilde{\mathbf{u}}_i\tilde{\mathbf{u}}_i^\top\} - \log\det\{\mathbf{\Lambda}_{\hat{\mathbf{x}}}^{-1}\} + \log\det\{\mathbf{\Lambda}_{\hat{\mathbf{x}}}^{-1} + \sum_{i\notin\mathcal{S}} \tilde{\mathbf{u}}_i\tilde{\mathbf{u}}_i^\top\} - \log\det\{\mathbf{\Lambda}_{\hat{\mathbf{x}}}^{-1}\}. \quad (5.1)$$

We denote the above cost function as $F(\mathcal{S}) = f(\mathcal{S}) + f(\mathcal{N}\backslash\mathcal{S})$, where $f(\mathcal{S}) = \log\det\{\mathbf{\Lambda}_{\hat{\mathbf{x}}}^{-1} + \sum_{i\in\mathcal{S}} \tilde{\mathbf{u}}_i\tilde{\mathbf{u}}_i^\top\} - \log\det\{\mathbf{\Lambda}_{\hat{\mathbf{x}}}^{-1}\}$. To solve this problem, we first propose the following theorem.

Theorem 3. *For any submodular function f, the function defined by $g(\mathcal{S}) = f(\mathcal{N} \setminus \mathcal{S})$ is submodular.*

Proof. Assume that $\mathcal{A} \subseteq \mathcal{B} \subseteq \mathcal{N}, k \notin \mathcal{B}$ and $k \in \mathcal{N}$, denote the set $\mathcal{N} \setminus (\mathcal{A} \cup \{k\})$ as $\mathcal{C}$, denote $\mathcal{N} \setminus \mathcal{A}$ as $\mathcal{C} \cup \{k\}$. Analogously, denote $\mathcal{N} \setminus (\mathcal{B} \cup \{k\})$ as $\mathcal{D}$, and $\mathcal{N} \setminus \mathcal{B}$ as $\mathcal{D} \cup \{k\}$. Obviously, $\mathcal{D} \subseteq \mathcal{C}$. Then by the definition of submodularity, we have

$$f(\mathcal{C}) - f(\mathcal{C} \cup \{k\}) \geqslant f(\mathcal{D}) - f(\mathcal{D} \cup \{k\}),$$

rearrange the terms we obtain

$$f(\mathcal{D} \cup \{k\}) - f(\mathcal{D}) \geqslant f(\mathcal{C} \cup \{k\}) - f(\mathcal{C}),$$

or equivalently,

$$f(\mathcal{N} \setminus (\mathcal{A} \cup \{k\})) - f(\mathcal{N} \setminus \mathcal{A}) \geqslant f(\mathcal{N} \setminus (\mathcal{B} \cup \{k\})) - f(\mathcal{N} \setminus \mathcal{B}),$$

replace f by g where $g(\mathcal{S}) = f(\mathcal{N} \setminus \mathcal{S})$, we have

$$g(\mathcal{A} \cup \{k\}) - g(\mathcal{A}) \geqslant g(\mathcal{B} \cup \{k\}) - g(\mathcal{B}),$$

which is exactly the definition of a submodular function; hence the proof is finished. □

Theorem 4. *Let $f(\mathcal{S})$ be a submodular function, and let $\mathcal{N}$ denote the ground set. Then function $F(\mathcal{S}) = f(\mathcal{S}) + f(\mathcal{N} \setminus \mathcal{S})$ is symmetric submodular.*

Proof. To see $F(\mathcal{S})$ is symmetric, we just have to show that $F(\mathcal{S}) = F(\mathcal{N} \setminus \mathcal{S})$. Indeed, $F(\mathcal{N} \setminus \mathcal{S}) = f(\mathcal{N} \setminus \mathcal{S}) + f(\mathcal{N} \setminus (\mathcal{N} \setminus \mathcal{S})) = f(\mathcal{N} \setminus \mathcal{S}) + f(\mathcal{S}) = F(\mathcal{S})$, which complemetes the first half of the proof.

Now we show $F(\mathcal{S})$ is submodular. Since $f(\mathcal{S})$ is submodular,

$$f(\mathcal{A} \cup \{k\}) - f(\mathcal{A}) \geqslant f(\mathcal{B} \cup \{k\}) - f(\mathcal{B}), \tag{5.2}$$

and according to theorem 4,

$$g(\mathcal{A} \cup \{k\}) - g(\mathcal{A}) \geqslant g(\mathcal{B} \cup \{k\}) - g(\mathcal{B}), \tag{5.3}$$

where $g(\mathcal{S}) = f(\mathcal{N} \setminus \mathcal{S})$. Take the sum of (5.2) and (5.3), we have

$$F(\mathcal{A} \cup \{k\}) - F(\mathcal{A}) \geqslant F(\mathcal{B} \cup \{k\}) - F(\mathcal{B}), \tag{5.4}$$

which implies that $F(\mathcal{S})$ is submodular. □

Therefore, the problem (5.1) is an unconstrained submodular maximization problem, which can be approximated by a *deterministic linear time* $1/2$*-approximation algorithm* efficiently [51], [52].

Algorithm 8 Two-Sided greedy algorithm for graph bi-partitioning

Input: ground set $\mathcal{N}$, $\mathcal{X}_0 = \varnothing, \mathcal{Y}_0 = \mathcal{N}$
Output: partition $\{\mathcal{S}_1, \mathcal{S}_2\}$

for $i \leftarrow 1$ to N **do**
  $a_i \leftarrow F(\mathcal{X}_{i-1} \cup v_i) - F(\mathcal{X}_{i-1})$
  $b_i \leftarrow F(\mathcal{Y}_{i-1} \setminus v_i) - F(\mathcal{Y}_{i-1})$
  if $a_i \geqslant b_i$ **then**
    $\mathcal{X}_i \leftarrow \mathcal{X}_{i-1} \cup v_i$
    $\mathcal{Y}_i \leftarrow \mathcal{Y}_{i-1}$
  else
    $\mathcal{X}_i \leftarrow \mathcal{X}_{i-1}$
    $\mathcal{Y}_i \leftarrow \mathcal{Y}_{i-1} \setminus v_i$
  end if
end for
$\mathcal{S}_1 \leftarrow \mathcal{X}_N$
$\mathcal{S}_2 \leftarrow \mathcal{N} \setminus \mathcal{X}_N$

Theorem 5. *Let $\mathcal{S}^*$ and $\mathcal{X}_N$ denote the optimal solution of the problem* (5.1) *and the solution obtained by Algorithm 8, respectively. Then $F(\mathcal{X}_N)/F(\mathcal{S}^*) \geqslant \frac{1}{2}$, where $F(\mathcal{S}) = \log\det\{\mathbf{\Lambda}_{\hat{\mathbf{x}}}^{-1} + \sum_{i\in\mathcal{S}} \tilde{\mathbf{u}}_i \tilde{\mathbf{u}}_i^\top\} - \log\det\{\mathbf{\Lambda}_{\hat{\mathbf{x}}}^{-1}\} + \log\det\{\mathbf{\Lambda}_{\hat{\mathbf{x}}}^{-1} + \sum_{i\notin\mathcal{S}} \tilde{\mathbf{u}}_i \tilde{\mathbf{u}}_i^\top\} - \log\det\{\mathbf{\Lambda}_{\hat{\mathbf{x}}}^{-1}\}$.*

Proof. See [52]. □

Algorithm 9 Recursive Two-Sided greedy algorithm for graph multiway partitioning

Input: number of subsets P, ground set $\mathcal{N}$, , $\mathcal{X}_0 = \varnothing, \mathcal{Y}_0 = \mathcal{N}$
Output: partition $\{\mathcal{S}_1, \cdots \mathcal{S}_P\}$
for $i \in 1$ to N **do**
  $a_i \leftarrow F(\mathcal{X}_{i-1} \cup v_i) - F(\mathcal{X}_{i-1})$
  $b_i \leftarrow F(\mathcal{Y}_{i-1} \setminus v_i) - F(\mathcal{Y}_{i-1})$
  if $a_i \geqslant b_i$ **then**
    $\mathcal{X}_i \leftarrow \mathcal{X}_{i-1} \cup v_i$
    $\mathcal{Y}_i \leftarrow \mathcal{Y}_{i-1}$
  else
    $\mathcal{X}_i \leftarrow \mathcal{X}_{i-1}$
    $\mathcal{Y}_i \leftarrow \mathcal{Y}_{i-1} \setminus v_i$
  end if
end for
$\mathcal{S}_1 \leftarrow \mathcal{X}_N$
$\mathcal{S}_2 \leftarrow \mathcal{N} \setminus \mathcal{X}_N$
$q = 2$, indicating the current number of subsets
while $q \leqslant P$ **do**
  $\mathcal{S}_t = \arg\max\ f(\mathcal{S}_j), j = 1, \cdots, q$
  run Two-Sided greedy on $\mathcal{S}_t$, and let $\mathcal{X}_t$ denote the output
  $\mathcal{S}_t \leftarrow \mathcal{X}_t$
  $\mathcal{S}_{q+1} \leftarrow \mathcal{N} \setminus \mathcal{X}_t$
  $q \leftarrow q + 1$
end while

5.1.2. Recursive Two-Sided greedy algorithm

For larger P, We propose a recursive partitioning algorithm based on the two-sided greedy algorithm shown above. Although there is no performance guarantee, numerical results in the following chapter show that this algorithm works fine, especially when P equals the powers of 2. The idea of the algorithm is this: we start by applying algorithm 8 to solve the problem (5.1), which results in 2 disjoint subsets $\mathcal{S}_1$ and $\mathcal{S}_2$ where $\mathcal{S}_1 \cup \mathcal{S}_2 = \mathcal{N}$. Then we view $\mathcal{S}_1$ and $\mathcal{S}_2$ as the two new ground sets and solve the (5.1) problem using algorithm 8, then for each subset, we can further bi-partition it into two disjoint subsets. We can keep partitioning until the number of subsets satisfies the requirement.

Note that although this algorithm can be applied to obtain a solution with any P value, it works best when $P = 2^t, \forall t \in \mathbb{Z}^+$, since in this case, the nodes for each subset are allocated in a relatively balanced way.

5.1.3. A $\frac{1}{2}$-approximation greedy algorithm

In [50], Lehmann et al. give an $\frac{1}{2}$-approximation using a simple greedy algorithm, where performance guarantee holds for any P. The idea is as follows: Initialize all subsets with empty. For each node, we always add it to the subset which benefits the most from adding it to the current set, i.e., the one with the maximal marginal gain. This algorithm is shown in 10.

Algorithm 10 Greedy Joint Partitioning

Input: ground set $\mathcal{N}$
Output: partition $\{\mathcal{B}_1, \cdots, \mathcal{B}_P\}$
$\mathcal{B}_i \leftarrow \varnothing, i = 1, \cdots, P$
for $i \in \mathcal{N}$ **do**
  $j \leftarrow \arg\max_p f(\mathcal{B}_p \cup \{i\}) - f(\mathcal{B}_p), \forall p = 1, \cdots, P$
  $\mathcal{S}_j \leftarrow \mathcal{S}_j \cup \{i\}$
end for

Theorem 6. *Let opt denote the cost value corresponding to the optimal solution of the problem* (4.1)*,*

$\{\mathcal{B}_1, \cdots, \mathcal{B}_P\}$ denote the partition obtained by Algorithm 10, then

$$\frac{\sum_{i=1}^{P}(\text{log det}\{\mathbf{\Lambda}_{\hat{\mathbf{x}}}^{-1} + \sum_{j\in\mathcal{D}_i}\tilde{\mathbf{u}}_j\tilde{\mathbf{u}}_j^\top\} - \text{log det}\{\mathbf{\Lambda}_{\hat{\mathbf{x}}}^{-1}\})}{opt} \geqslant \frac{1}{2}. \tag{5.5}$$

Proof. See [50]. □

5.2. Randomized rounding approaches

In the node selection problem, besides the greedy algorithm, we can also obtain a discrete approximation by thresholding the solution of the convex relaxation of the original problem. So it is natural to think of the partitioning problem similarly. The convex relaxation of the problem (4.1) is given by

$$\begin{aligned} &\underset{y,\mathbf{Z}\in[0,1]^{N\times P}}{\text{maximize}} \sum_{p=1}^{P}(\text{log det}\left\{\left(\mathbf{U}_K^\top \text{diag}\left\{\mathbf{z}_p\right\}\mathbf{U}_K + \mathbf{\Lambda}_{\boldsymbol{x}}^{-\mathbf{1}}\right)\right\} - \text{log det}\{\mathbf{\Lambda}_{\boldsymbol{x}}^{-\mathbf{1}}\}) \\ &\text{subject to } \sum_{p=1}^{P}\mathbf{z}_p = \mathbf{1}, \end{aligned} \tag{5.6}$$

The solution is given by $[\mathbf{Z}^*]_{i,j} = \frac{1}{P}, \forall i,j = 1,\cdots,N$, where $\mathbf{Z} \in \mathbb{R}^{N\times K}$ is the partition indicator matrix as we defined in Chapter 4. Remark: Although the convex optimization problem shown in (5.6) is different from the one in Chapter 4 ((4.5)), the optimal solutions for both problems are the same. As discussed in Chapter 4, getting a binary solution from this balanced solution is impossible; something useful still exists. The first is that we get an upper bound for the original problem (4.1), which is presented in proposition 2.

Proposition 2. *Let opt denote the optimal value of the problem* (4.1)*, then $opt \leqslant P\,\text{log det}\{\mathbf{\Lambda}_{\hat{\mathbf{x}}}^{-1} + \frac{1}{P}\mathbf{I}\}$.*

Proof. The proof is trial since the solution of the original set problem must be within the feasible set of the relaxed convex problem, so opt must be less or equal to the optimal value of the convex problem. □

The second is that the cost function is maximized when the nodes are distributed to each representative subset in a balanced way. By the term balance, we mean not only the cost values for all subsets, which are $f(\mathcal{S}_p) = \text{log det}\{\mathbf{\Lambda}_{\hat{\mathbf{x}}}^{-1} + \sum_{i\in\mathcal{S}_p}\tilde{\mathbf{u}}_i\tilde{\mathbf{u}}_i^\top\} - \text{log det}\{\mathbf{\Lambda}_{\hat{\mathbf{x}}}^{-1}\}, p = 1,\cdots,P$, are close, but the Frobenius norm $\|\sum_{i\in\mathcal{S}_m}\tilde{\mathbf{u}}_i\tilde{\mathbf{u}}_i^\top - \sum_{i\in\mathcal{S}_n}\tilde{\mathbf{u}}_i\tilde{\mathbf{u}}_i^\top\|_F, \forall m,n \in \{1,\cdots,P\}, m \neq n$ is small as well. Now the question is, how do we promote the balance while keeping the solution binary? A reasonable way is to treat the rows of Z matrix as a probability distribution for each node because all the rows of $\mathbf{Z}$ sum up to one, and then use some randomized rounding techniques to make the solution balanced in expectation.

With the above analysis, the randomized strategy is this: for each node j, we allocate it to each subset with equal probability $\frac{1}{P}$. However, there are many ways to achieve this; which is a good randomized rounding strategy? [53] shows that if only value queries are available, then approximating the Submodular Welfare problem within a ratio strictly better than $1 - \frac{1}{e}$ is NP-hard. In this section, we introduce some well-designed randomized algorithms that can approach this bound when N gets larger and larger.

5.2.1. Simple randomized rounding

The first algorithm we propose is this: Assume that a special dice has P faces, and the probability of landing facing up for each face is the same, i.e., $\frac{1}{P}$. Embed each subset with each face of the dice, and roll the dice for each node to decide which subset it is assigned to. Let $\mathcal{D}_i$ denote the i-th (random) subset obtained in this way. For all $i = 1,\cdots,P$, the distributions of $\mathcal{D}_i$ are identical because every node can be assigned to every subset with equal probability $\frac{1}{P}$, which promotes the balance in the sense of taking expectation. This algorithm is presented in Algorithm 11. In parallel, [52] proves there exists a $1 - \left(1 - \frac{1}{P}\right)^{P-1}$-approximation factor for Algorithm 11, as we show in Theorem 7.

Algorithm 11 Simple randomized rounding Partitioning

Input: ground set N
Output: Partition $\{\mathcal{D}_1, \cdots, \mathcal{D}_P\}$
1: $\mathcal{D}_i \leftarrow \varnothing, i = 1, \cdots, P$
2: **for** every node $v \in \mathcal{N}$ **do**
3: Randomly select an element i from the set $\{1, \cdots, P\}$ with identical probability $\frac{1}{P}$
4: $\mathcal{D}_i \leftarrow \mathcal{D}_i \cup v$
5: **end for**

Theorem 7. *Let opt denote the cost function value corresponding to the optimal solution of the problem* (4.1)*, $\{\mathcal{D}_1, \cdots, \mathcal{D}_P\}$ denote the partition obtained by Algorithm 11, then*

$$\frac{\mathbb{E}\left[\sum_{i=1}^{P}(\log\det\{\mathbf{\Lambda}_{\hat{\mathbf{x}}}^{-1} + \sum_{j\in\mathcal{D}_i}\tilde{\mathbf{u}}_j\tilde{\mathbf{u}}_j^{\top}\} - \log\det\{\mathbf{\Lambda}_{\hat{\mathbf{x}}}^{-1}\})\right]}{opt} \geqslant 1 - \left(1 - \frac{1}{P}\right)^{P-1} \tag{5.7}$$

Proof. See [52]. □

Observe theorem 7, when P goes to infinity, the performance bound goes to $1 - \frac{1}{e}$.

5.2.2. Contention Resolution rounding

This chapter introduces a modified algorithm based on the fair contention resolution algorithm proposed in [49], [54]. With a balanced solution, the algorithm runs as follows: each subset samples a random subset independently, where each node is added with probability $\frac{1}{P}$. Thus, we obtain P random subsets, and with a high probability that disputed nodes exist in the sense that these nodes are added to more than one subset. Then, we randomly assign these disputed nodes to one of the subsets with probability $\frac{1}{P}$. In [49], it can be shown that this algorithm can achieve a $1-(1-\frac{1}{P})^P$ approximation ratio. However, this algorithm appears to have a considerable amount of slackness since many nodes may not be added to any subset. Therefore, we add another stage to this algorithm: for nodes not allocated in the first two rounds, each node is randomly added to one of the subsets with probability $\frac{1}{P}$. Although doing so does not improve the approximation ratio mentioned above, it gives better results than the original algorithm (can never be worse) because f is monotone; adding any node to a subset can never decrease its utility function value. The detailed algorithm is given in the following.

Algorithm 12 Contention Resolution rounding Partitioning

Input: ground set N
Output: Partition $\{\mathcal{F}_1, \cdots, \mathcal{F}_P\}$
1: $\mathcal{F}_i \leftarrow \varnothing, i = 1, \cdots, P$
2: Each subset $\mathcal{F}_i, \forall i = 1, \cdots, P$ samples a random subset of N where each node appears independently with probability $\frac{1}{P}$.
3: $\mathcal{C} \leftarrow \bigcup \mathcal{F}_i \cap \mathcal{F}_j, \forall i, j$.
4: **for** $i \in \mathcal{C}$ **do**
5: $\mathcal{R}_i \leftarrow$ set of labels of subsets that request node i
6: randomly add i to one of the subset in $\mathcal{R}_i$ with probability $\frac{1}{|\mathcal{R}_i|}$
7: **end for**
8: **for** $i \in \mathcal{N} \setminus \bigcup \mathcal{F}_i, \forall i$ **do**
9: add node i randomly to $\mathcal{F}_i, \forall i$ with probability $\frac{1}{P}$
10: **end for**

Theorem 8. *Let opt denote the cost function value corresponding to the optimal solution of the problem* (4.1)*, $\{\mathcal{F}_1, \cdots, \mathcal{F}_P\}$ denote the partition obtained by Algorithm 12, then*

$$\frac{\mathbb{E}\left[\sum_{i=1}^{P}(\log\det\{\mathbf{\Lambda}_{\hat{\mathbf{x}}}^{-1} + \sum_{j\in\mathcal{D}_i}\tilde{\mathbf{u}}_j\tilde{\mathbf{u}}_j^{\top}\} - \log\det\{\mathbf{\Lambda}_{\hat{\mathbf{x}}}^{-1}\})\right]}{opt} \geqslant 1 - \left(1 - \frac{1}{P}\right)^{P} \tag{5.8}$$

Proof. The proof of the approximation ratio for the original 2-stage rounding algorithm can be found in [49]. To see this also holds for our modified algorithm is trial. The only difference is the last rounding stage, where nodes that are not allocated are added to each subset. Since the utility function for each subset is monotone, the function value can never decrease when these nodes are added; hence the performance of Algorithm 12 is at least as good as the 2-stage rounding algorithm. □

Though the performance guarantee achieved by this rounding technique differs from the previous one, the limit of the approximation ratio is $\frac{1}{e}$ as well when P goes to infinity.

6

Distributed Partitioning Framework

The algorithms we described in the last chapter have linear time complexity in N, which is quite efficient. However, running such algorithms becomes impractical when N gets extremely large. To handle the massive data, several distributed algorithms for submodular maximization have been proposed recently [55], [56], [57], where a single representative subset is found. In this work, We propose a distributed framework for the partitioning problem where disjoint representative subsets can be obtained by paralyzing some serial algorithms, including SIP, Recursive Two-Sided greedy, and randomized rounding algorithms.

The MapReduce style computing model [58] has proven successful in many large-scale machine learning and data mining algorithms [59], [60], [55]. The general idea of MapReduce that distributing the data to independent processors, then all processors can process their allocated data in parallel. Based on this idea, we propose a distributed graph partitioning framework. Likewise, we distribute all nodes to m processors such that each node is assigned to one of the processors. Then each machine partitions its allocated nodes using algorithms shown in the previous chapters, so we obtain m local partitions. In the last step, we merge all location partitions obtained by the m processors to get the global partition back. This process is illustrated in Fig. 6.1, and the details are shown in Algorithm 13.

Notations in the remaining of this chapter: denote by $\mathcal{S}_p^d$ the p-th representative subset obtained by the distributed algorithm, $\mathcal{V}_i$ the set of nodes allocated to the i-th machine, $\mathcal{S}_p^{(i)}$ the p-th local representative subset corresponding to the i-th machine. When we refer to function f, we always assume that $f(\mathcal{S}) = \text{log det}\{\mathbf{\Lambda}_{\hat{\mathbf{x}}}^{-1} + \sum_{i\in\mathcal{S}} \tilde{\mathbf{u}}_i\tilde{\mathbf{u}}_i^\top\} - \text{log det}\{\mathbf{\Lambda}_{\hat{\mathbf{x}}}^{-1}\}$ as previous. Let $opt(m)$ denote the optimal value of the problem

$$\begin{array}{rl} \text{maximize} & f\left(\mathcal{S}_1\right) + \cdots + f\left(\mathcal{S}_m\right) \\ \text{subject to} & \mathcal{S}_1 \cup \cdots \cup \mathcal{S}_m = \mathcal{N} \\ & \mathcal{S}_i \cap \mathcal{S}_j = \varnothing, \forall i, j, \end{array} \tag{6.1}$$

and indeed, $opt(m) = opt$ as we defined in the last chapter when m=P.

Algorithm 13 Distributed node partitioning

Input: ground set $\mathcal{N}$, number of processors m, number of subsets P
Output: partition $\{\mathcal{S}_1^d, \cdots \mathcal{S}_P^d\}$

1: Phase 1: Distributed all nodes into m sets $\mathcal{V}_1, \cdots, \mathcal{V}_m$, each corresponds to a machine
2: **for** $i \in \{1, \cdots, m\}$ **do in parallel**
3: Phase 2: run some partitioning algorithms, output $\{\mathcal{S}_1^{(i)}, \cdots \mathcal{S}_P^{(i)}\}$
4: **end for**
5: Phase 3: Merge the partitions obtained by all the processors: $\mathcal{S}_p^d \leftarrow \mathcal{S}_p^{(1)} \cup \mathcal{S}_p^{(2)} \cup \cdots \cup \mathcal{S}_p^{(m)}, \forall p = 1, \cdots, P$

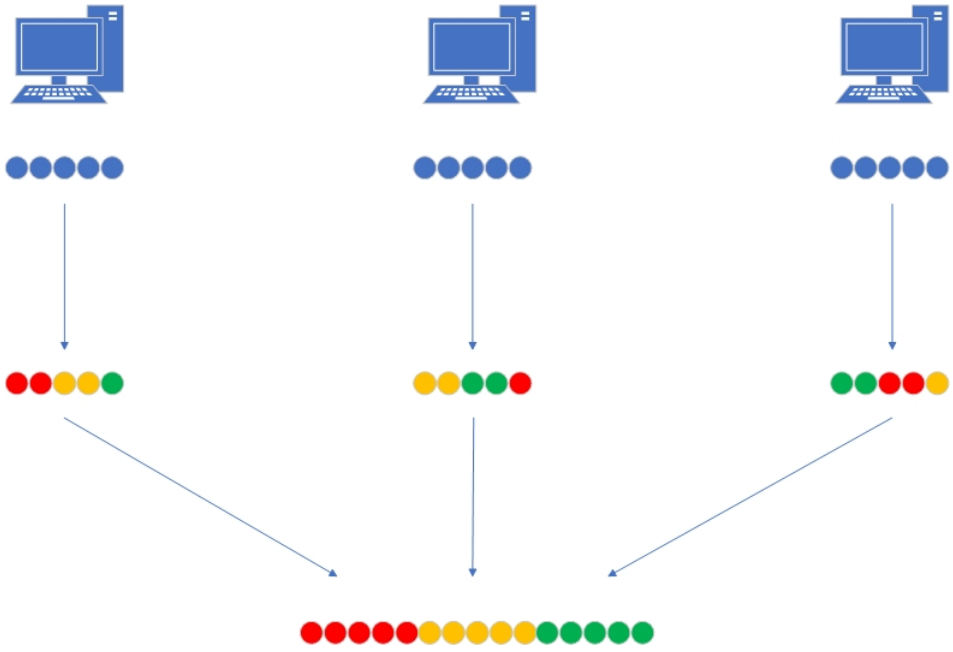

Figure 6.1: An illustration of the distributed partitioning framework with three processors and 15 nodes. The color of the node indicates which subset they are assigned to.

6.1. Performance Analysis

In this section, we derive two lower bounds of Algorithm 13 when the nodes are distributed in the manner of Algorithm 11 and Algorithm 12. We first introduce the following lemma.

Lemma 2. *For any disjoint subsets* $\mathcal{S}_1, \mathcal{S}_2, \cdots, \mathcal{S}_l$,

$$f(\mathcal{S}_1 \cup \cdots \cup \mathcal{S}_l) \leqslant f(\mathcal{S}_1) + \cdots + f(\mathcal{S}_l), \forall l. \tag{6.2}$$

Proof. Denote $f(\mathcal{A} \mid \mathcal{B}) = f(\mathcal{A} \cup \mathcal{B}) - f(\mathcal{B})$. Using submodularity,

$$f(\mathcal{A}) + f(\mathcal{B}) \geqslant f(\mathcal{A} \cup \mathcal{B}) + f(\mathcal{A} \cap \mathcal{B}). \tag{6.3}$$

Since f is normalized, when $\mathcal{A}$ and $\mathcal{B}$ are disjoint, $f(\mathcal{A} \cap \mathcal{B}) = 0$. Thus,

$$f(\mathcal{A} \mid \mathcal{B}) \leqslant f(\mathcal{A}). \tag{6.4}$$

Using telescoping sum, we have

$$f(\mathcal{S}_1 \cup \cdots \cup \mathcal{S}_l) = f(\mathcal{S}_1) + f(\mathcal{S}_2 \mid \mathcal{S}_1) + \cdots + f(\mathcal{S}_l \mid \mathcal{S}_{l-1} \cup \cdots \cup \mathcal{S}_1), \tag{6.5}$$

combine (6.4) and (6.5), we see that Lemma 6 holds. □

The following theorem and corollary are based on the result from theorem 7.

Theorem 9. *If the nodes are distributed to the* m *processors* $\mathcal{V}_1, \cdots, \mathcal{V}_m$ *in the manner as Algorithm 11 in Phase 1, then for Algorithm 13, and regardless of how Phase 2 and 3 are implemented, for any output* $\{\mathcal{S}_1^d, \cdots, \mathcal{S}_P^d\}$*, we always have*

$$\mathbb{E}\left[\sum_{p=1}^{P} f(\mathcal{S}_p^d)\right] \geqslant \left[1 - \left(1 - \frac{1}{m}\right)^{m-1}\right] \cdot \frac{opt(m)}{m}. \tag{6.6}$$

Proof. By lemma 6, for any partition $\{\mathcal{S}_1^{(i)}, \cdots, \mathcal{S}_P^{(i)}\}$ of $\mathcal{V}_i$, we have

$$\begin{aligned} f(\mathcal{S}_1^{(i)}) + f(\mathcal{S}_2^{(i)}) + \cdots + f(\mathcal{S}_P^{(i)}) &\geqslant f(\mathcal{S}_1^{(i)} \cup \mathcal{S}_2^{(i)} \cup \cdots \cup \mathcal{S}_P^{(i)}) \\ &= f(\mathcal{V}_i), \forall i = 1, \cdots, m. \end{aligned} \tag{6.7}$$

Since f is monotone,

$$f(\mathcal{S}_p^{(1)} \cup \mathcal{S}_p^{(2)} \cup \cdots \cup \mathcal{S}_p^{(m)}) \geqslant f(\mathcal{S}_p^{(i)}), \forall i = 1, \cdots, m, p = 1, \cdots, P. \tag{6.8}$$

Therefore,

$$\begin{aligned} \sum_{p=1}^{P} f(\mathcal{S}_p^d) &= \sum_{p=1}^{P} f(\mathcal{S}_p^{(1)} \cup \mathcal{S}_p^{(2)} \cup \cdots \cup \mathcal{S}_p^{(m)}) \geqslant f(\mathcal{S}_1^{(i)}) + f(\mathcal{S}_2^{(i)}) + \cdots + f(\mathcal{S}_P^{(i)}) \\ &\geqslant f(\mathcal{S}_1^{(i)} \cup \mathcal{S}_2^{(i)} \cup \cdots \cup \mathcal{S}_P^{(i)}) = f(\mathcal{V}_i), \forall i = 1, \cdots, m. \end{aligned} \tag{6.9}$$

Take expectation on both sides,

$$\mathbb{E}[\sum_{p=1}^{P} f(\mathcal{S}_p^d)] \geqslant \mathbb{E}[f(\mathcal{V}_i)]. \tag{6.10}$$

Because every node in $\mathcal{V}_i$ are independently assigned with probability $\frac{1}{m}$, by Theorem 7, we have

$$\mathbb{E}\left[\sum_{p=1}^{P} f(\mathcal{S}_p^d)\right] \geqslant \left[1 - \left(1 - \frac{1}{m}\right)^{m-1}\right] \cdot \frac{opt(m)}{m}. \tag{6.11}$$

□

Corollary 10. *If $m \geqslant P$ and the nodes are distributed to the m processors $\mathcal{V}_1, \cdots, \mathcal{V}_m$ the manner as Algorithm 11 in Phase 1, then for Algorithm 13, and regardless of how Phase 2 and 3 are implemented, for any output $\{\mathcal{S}_1^d, \cdots, \mathcal{S}_P^d\}$, we always have*

$$\mathbb{E}\left[\sum_{p=1}^{P} f(\mathcal{S}_p^d)\right] \geqslant \left[1 - \left(1 - \frac{1}{m}\right)^{m-1}\right] \cdot \frac{opt}{m}. \tag{6.12}$$

Proof. We have to prove that when $m \geqslant P$, $opt(m) \geqslant opt$. Without loss of generality, we assume $m = P + 1$. Denote the optimal solution w.r.t. $opt(m)$ by $\{\mathcal{V}_1^*, \cdots, \mathcal{V}_m^*\}$, and the optimal solution w.r.t. opt by $\{\mathcal{S}_1^*, \cdots, \mathcal{S}_P^*\}$. For an arbitrary $j = 1, \cdots, P$, we partition $\mathcal{S}_j^*$ into $\mathcal{S}_{j1}$ and $\mathcal{S}_{j2}$. By lemma 6.2 we have $f(\mathcal{S}_{j1}) + f(\mathcal{S}_{j2}) \geqslant f(\mathcal{S}_j^*)$ for any partition $\{\mathcal{S}_{j1}, \mathcal{S}_{j2}\}$. Therefore,

$$f(\mathcal{S}_1^*) + \cdots + f(\mathcal{S}_{j1}) + f(\mathcal{S}_{j2}) + \cdots + f(\mathcal{S}_P^*) \geqslant f(\mathcal{S}_1^*) + \cdots + f(\mathcal{S}_j^*) + \cdots + f(\mathcal{S}_P^*) = opt. \tag{6.13}$$

Since $opt(m)$ is the optimal when the number of subsets is m, hence we have

$$opt(m) = f(\mathcal{V}_1^*) + \cdots + f(\mathcal{V}_m^*) \geqslant f(\mathcal{S}_1^*) + \cdots + f(\mathcal{S}_{j1}) + f(\mathcal{S}_{j2}) + \cdots + f(\mathcal{S}_P^*). \tag{6.14}$$

Combining (6.13) and (6.14), we see the corollary holds. □

Analogously, we have the following theorem and corollary, based on the result from theorem 8. Here we omit the proofs since the same procedures are used.

Theorem 11. *If the nodes are distributed to the m processors $\mathcal{V}_1, \cdots, \mathcal{V}_m$ in the manner as Algorithm 12 in Phase 1, then for Algorithm 13, and regardless of how Phase 2 and 3 are implemented, for any output $\{\mathcal{S}_1^d, \cdots, \mathcal{S}_P^d\}$, we always have*

$$\mathbb{E}\left[\sum_{p=1}^{P} f(\mathcal{S}_p^d)\right] \geqslant \left[1 - \left(1 - \frac{1}{m}\right)^{m}\right] \cdot \frac{opt(m)}{m}. \tag{6.15}$$

Corollary 12. *If $m \geqslant P$ and the nodes are distributed to the m processors $\mathcal{V}_1, \cdots, \mathcal{V}_m$ the manner as Algorithm 12 in Phase 1, then for Algorithm 13, and regardless of how Phase 2 and 3 are implemented, for any output $\{\mathcal{S}_1^d, \cdots, \mathcal{S}_P^d\}$, we always have*

$$\mathbb{E}\left[\sum_{p=1}^{P} f(\mathcal{S}_p^d)\right] \geqslant \left[1 - \left(1 - \frac{1}{m}\right)^{m}\right] \cdot \frac{opt}{m}. \tag{6.16}$$

The above performance bounds hold for any partitioning algorithm under certain conditions. However, for Algorithm 11, the centralized bound also holds in the distributed case regardless of the manner of node assignment in step 1 of Algorithm 13. This is trivial because no matter how we allocate nodes to processors, every node is added to each representative subset independently with equal probability $\frac{1}{P}$. Thus the distribution of every representative subset obtained by the distributed version of Algorithm 11 remains precisely the same as the one obtained by the centralized one.

Theorem 13. *If Algorithm 11 is used in Phase 3 of the distributed node partitioning framework, and let* $\{\mathcal{D}_1^d, \cdots, \mathcal{D}_P^d\}$ *denote the output, regardless of how Phase 1 and Phase 2 are implemented, we always have*

$$\frac{\mathbb{E}\left[\sum_{i=1}^{P}(\log\det\{\mathbf{\Lambda}_{\hat{\mathbf{x}}}^{-1}+\sum_{j\in\mathcal{D}_i^d}\tilde{\mathbf{u}}_j\tilde{\mathbf{u}}_j^\top\}-\log\det\{\mathbf{\Lambda}_{\hat{\mathbf{x}}}^{-1}\})\right]}{opt}\geqslant 1-\left(1-\frac{1}{P}\right)^{P-1} \tag{6.17}$$

7

Results and Discussion

In this chapter, we run presented algorithms and compare their performance on synthetic and real-world data. For synthetic data, four types of graphs are generated (random sensor network, Minnesota road graph, graphs produced by stochastic block model, and Erdos-Renyi graphs). Though the original SIP and JIP algorithm are based on A-optimal design, extending them to D-optimal design is trivial. In the remainder of this chapter, SIP and JIP we used are based on D-optimal design. Moreover, we normalize the cost function by the factor $P(\log\det\{\mathbf{\Lambda}_{\hat{\mathbf{x}}}^{-1}+\frac{1}{P}\mathbf{I}\}-\log\det\{\mathbf{\Lambda}_{\hat{\mathbf{x}}}^{-1}\})$ to force its range be $[0,1]$, as by proposition 2, $P(\log\det\{\mathbf{\Lambda}_{\hat{\mathbf{x}}}^{-1}+\frac{1}{P}\mathbf{I}\}-\log\det\{\mathbf{\Lambda}_{\hat{\mathbf{x}}}^{-1}\})$ is an upper bound of the problem 4.1.

7.1. Synthetic Dataset

7.1.1. Setup

For the remaining of this chapter, we have the following settings: the noise covariance matrix $\mathbf{\Lambda}_{\boldsymbol{w}}=\mathbf{I}$, and $\mathbf{\Lambda}_{\hat{\mathbf{x}}}=\mathbb{E}\hat{\mathbf{x}}_K\hat{\mathbf{x}}_K^\top=\sigma_x\mathbf{I}$, where $\sigma_x=2$. Different values are set in different scenarios for the bandwidth K, the number of nodes N, and the number of subsets P.

7.1.2. Simulation results

JIP Vs. SIP

Among all these algorithms, JIP is a special one since P is not an input but the value it aims to maximize. Therefore, we first compare it with SIP in this way: fix P and run SIP, set ϵ to be the function value corresponding to the worst group, and use this ϵ to run JIP. Denote the number of disjoint subsets P', we compare P and P' to see which algorithm performs better.

Sensor network We start with the sensor network from David generated by the GSPBOX toolbox, where $N=64$ and K are set to 5. The graph is shown in Fig. 7.1.

We see that when $P=2,3,4,5,7$, P' has the same values. Therefore, we compare these cases' (normalized) cost values.

value of P	value of P'
2	2
3	3
4	4
5	5
6	5
7	7
8	7

Table 7.1: Performance of SIP and JIP on random sensor graph regarding the number of subsets.

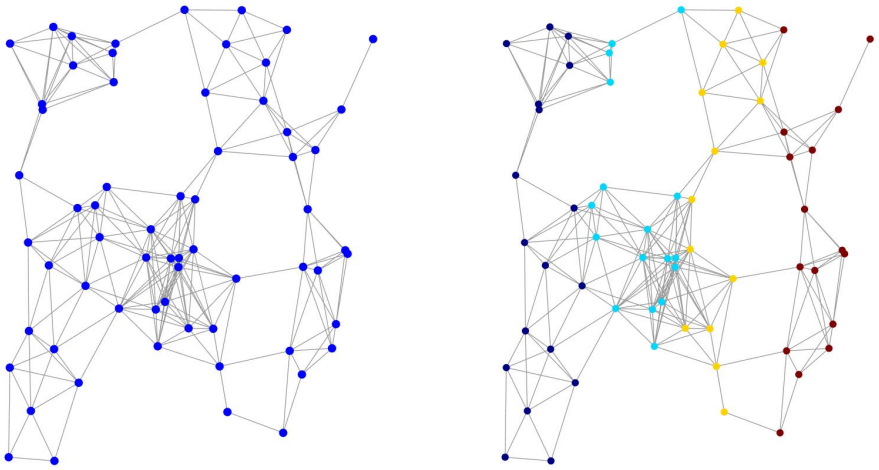

Figure 7.1: Sensor network from David(right) and the corresponding distribution of processors.

	SIP's cost	JIP's cost
$P/P' = 2$	0.9858	0.9716
$P/P' = 3$	0.9780	0.9505
$P/P' = 4$	0.9697	0.9343
$P/P' = 5$	0.9584	0.9271
$P/P' = 7$	0.9085	0.9223

Table 7.2: Performance of SIP and JIP on random sensor graph regarding the cost function value.

Minnesota road graph We proceed with a Minnesota road graph where $N = 2642$ and $K = 10$. The graph is shown in Fig. 7.2.

Results of the other algorithms

Here we show the results of algorithms given in Chapter 5, along with SIP as a baseline. The graphs above are used here as well. For the distributed version of these algorithms, we let $m = 4$ and distribute nodes to each processor uniformly randomly and fix this allocation for each graph. Since there are randomized algorithms, we run them 1000 times on a graph, calculate the mean and standard deviation of all their realizations, and denote them as Mean $\pm$ SD in the following tables. To show the robustness of these algorithms over different graphs, 100 realizations of some random graphs are generated, e.g., graphs with the stochastic block model, and likewise, we compute the mean and the standard deviation for all algorithms.

Sensor network We start with a random sensor network from David with $N = 64$ and $K = 5$. The results for $P = 5$ are shown in Table 7.4.

value of P	value of P'
2	1
3	2
4	3
5	4
6	5
7	6
8	7

Table 7.3: Performance of SIP and JIP on Minnesota road graph regarding the number of subsets.

Figure 7.2: Minnesota road graph(left) and the corresponding distribution of processors(right).

algorithm	cost
SIP	0.7761
Double greedy	0.9563
Simple greedy	0.9699
Simple randomized	0.9314 $\pm$ 0.0113
Contention resolution	0.9312 $\pm$ 0.0112
Distributed SIP	0.9457
Distributed double greedy	0.9570
Distributed simple greedy	0.9641
Distributed simple randomized	0.9314 $\pm$ 0.0115
Distributed contention resolution	0.9324 $\pm$ 0.0109

Table 7.4: Performance of partitioning algorithms on the sensor network from David with $P = 5$.

algorithm	cost
SIP	0.6634 ± 0.0436
Double greedy	0.8980 ± 0.0176
Simple greedy	0.9077 ± 0.0183
Simple randomized	0.8769 ± 0.0163
Contention resolution	0.8768 ± 0.0163
Distributed SIP	0.8753 ± 0.0192
Distributed double greedy	0.8877 ± 0.0172
Distributed simple greedy	0.8917 ± 0.0172
Distributed simple randomized	0.8769 ± 0.0163
Distributed contention resolution	0.8769 ± 0.0163

Table 7.5: Performance of partitioning algorithms on stochastic block model with $P = 5$.

algorithm	cost
SIP	0.5736 ± 0.0687
Double greedy	0.8492 ± 0.0359
Simple greedy	0.8573 ± 0.0378
Simple randomized	0.8275 ± 0.0327
Contention resolution	0.8274 ± 0.0327
Distributed SIP	0.8231 ± 0.0356
Distributed double greedy	0.8321 ± 0.0342
Distributed simple greedy	0.8380 ± 0.0351
Distributed simple randomized	0.8274 ± 0.0327
Distributed contention resolution	0.8274 ± 0.0326

Table 7.6: Performance of partitioning algorithms on Erdos-Reyni graphs $P = 5$.

Graphs produced by stochastic block model Here we produce random graphs using the stochastic block model by the GSPBOX toolbox with $N = 100$. The parameter of the graph is as follows: the intra-cluster edge probability is set to 0.7 while the inter-cluster edge probability is set to 0.3, and the cluster number is set to 5. The bandwidth of the graph signal is set to $K = 10$. The results for $P = 5$ are shown in Table 7.5.

Erdos-Renyi graphs In this section, we test our algorithms on Erdos-Reyni graphs with $N = 100$. The parameter of the graph is as follows: the probability of connection of a node with another is set to 0.1. The bandwidth of the graph signal is set to $K = 10$. The results of $P = 5$ are shown in Table 7.6.

7.2. French National Meteorological Dataset

In this section, we apply our algorithms to a humidity measurements dataset published by the French national meteorological with hourly humidity measurements (744 time instances in total) on 22 weather stations collected during with hourly humidity measurements (744 time instances in total) on 22 weather stations collected during January 2014 in the region of Brest. This dataset is also used in [33].

7.2.1. Graph Construction

We construct the graph based on KNN as the authors did in [33]: Considering a weather station as a node and connecting the five nearest stations with the weight function $w(i, n) = e^{-d_{i,n}^2 \tau}$, where $d_{i,n}$ is the euclidean distance between station i and n and τ is a tuning parameter. For our experiments, we consider every time instant as an independent realization of a stationary graph signal.

7.2.2. Data Preprocessing

We first remove the humidity mean of each station independently. Then we remove the higher frequency components of every time instance to force the graph signals for all frames to have a bandwidth $K = 4$. We stack all the measurements into a $N \times F$ matrix $\mathbf{X}$ where $N = 22$ and $F = 724$. Then the covariance

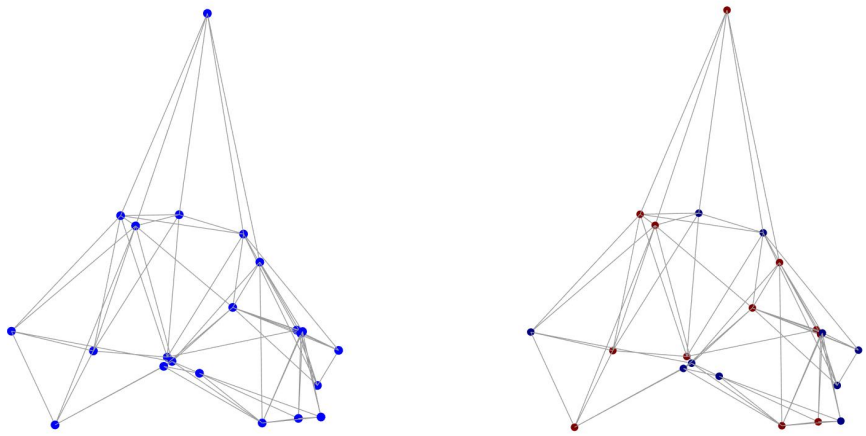

Figure 7.3: The Brest weather stations graph(left) and its distribution of processors(right).

matrix of the signal can be estimated as $\hat{\mathbf{\Lambda}}_{\mathbf{x}} = \frac{1}{F-1}\mathbf{X}\mathbf{X}^{\top}$. The Fourier transform of the covariance is thus estimated by $\hat{\mathbf{\Lambda}}_{\hat{\mathbf{x}}} = \mathbf{U}^{\top}\hat{\mathbf{\Lambda}}_{\mathbf{x}}\mathbf{U}$. Then we add AWGN to $\mathbf{X}$ and denote by $\mathbf{Y}$ the noisy observation of $\mathbf{X}$, i.e., $\mathbf{Y} = \mathbf{X} + \mathbf{N}$, where the noise power is set to one.

7.2.3. Results

Parameters: number of subsets $P = 2$, number of processors $m = 2$, the bandwidth $K = 4$, and the nodes are distributed to processors in an i.i.d manner. We show the results in Table 7.7 in terms of both the D-optimal based cost value and the normalized Rooted Mean Square error (NRMSE), which we defined as: NRMSE$= \frac{\frac{1}{P}\sum_{p=1}^{P}\left\|\mathbf{X}'_p - \mathbf{X}\right\|_F}{\|\mathbf{X}\|_F}$, where $\mathbf{X}'_p$ denotes the reconstructed data corresponding to the p-th representative sampling subset obtained by an partitioning algorithm. In Table 7.7, we also include results of four types of exhaustive search. Here we explain what they stand for. Exhaustive searches 1 and 2 are based on the D-optimal cost function, while Exhaustive searches 3 and 4 are based on NRMSE. Exhaustive search 1 shows the result of double the cost corresponding to the optimal subset with eleven nodes; Exhaustive search 2 shows the result of the partition where the first subset is the optimal subset with eleven nodes and the second subset is the complement set. Similar to 1 and 2, Exhaustive searches 3 and 4 show the results in terms of the NRMSE.

7.3. Discussion

Comparing SIP with JIP, from Table 7.1 and 7.3, we see that for a given P, the corresponding P' is always smaller or equal to P. From Table 7.2, except when $P = P' = 7$, SIP always performs better. Hence, SIP is a more advanced algorithm compared to JIP. That is why we consider SIP as the current baseline.

We first focus on centralized algorithms. From a performance perspective, Simple greedy is the best algorithm for synthetic datasets. The performance of Double greedy is close yet slightly worse. The two randomized algorithms perform a bit worse than double greedy but still work fine for their large mean and relatively small standard deviation. Moreover, their performance is close. SIP is the most inferior centralized algorithm among the five. For the real-world dataset, the performance situation remains almost the same in terms of D-optimal cost. However, regarding NRMSE, the SIP exceeds the two randomized algorithms.

For distributed algorithms, in the synthetic scenario, though not precisely the same, the performance for each algorithm remains compared to their centralized counterpart in general, except SIP, whose performance improves significantly. For the real-world dataset, the distributed framework decreases the performance of these algorithms but at an acceptable level.

algorithm	Cost	NRMSE
SIP	0.7610	0.1032
Simple greedy	0.9642	0.0987
Simple randomized	0.9175 $\pm$ 0.0395	0.1292$\pm$ 0.0290
Contention resolution	0.9169 $\pm$ 0.0395	0.1296$\pm$ 0.0290
Double greedy	0.9642	0.0987
Distributed SIP	0.8804	0.1494
Distributed double greedy	0.8909	0.1508
Distributed simple greedy	0.9591	0.0999
Distributed simple randomized	0.9174 $\pm$ 0.0394	0.1294$\pm$ 0.0293
Distributed contention resolution	0.9171 $\pm$ 0.0397	0.1296$\pm$ 0.0293
Exhaustive search 1	1.0857	
Exhaustive search 2	0.8883	
Exhaustive search 3		0.0727
Exhaustive search 4		0.1132

Table 7.7: Performance of partitioning algorithms on French National Meteorological dataset.

An interesting fact about the two randomized algorithms and their distributed extensions is that although they have different performance bounds, their performance is almost identical.

From the results of four exhaustive searches and the fact that the cost value is close to one, we conclude that our proposed algorithms are near-optimal.

Here we give some guidance about which algorithm to apply in different circumstances. Note that although the two greedy approaches have higher performance, their complexity is higher since they require the valuation of submodular functions, which requires polynomial time in K. On the other hand, though the randomized algorithms perform worse, their complexity is just linear in N ($\mathcal{O}(N)$) since they do not have to access the objective function explicitly. So the key to applying the right algorithm is a performance-efficiency trade-off, and here is the strategy we suggest:

- When only one central computer is available, if the graph size is large, run a randomized algorithm r times and pick the best realization. The larger the graph is, the smaller r needs to be set. If the graph size is small, then the two greedy algorithms are recommended for better performance.
- When there are m computers available, if $\frac{N}{m}$ is small, meaning that each local computer needs only access a small fraction of nodes, then two distributed greedy algorithms can be applied. The distributed randomized algorithms are better if $\frac{N}{m}$ is large.

8

Concluding Remarks

8.1. Conclusion

In this work, based on the graph node selection problem, we formulate the graph node partitioning problem. Then we show that our problem formulation is a particular case of a Submodular Welfare problem. Besides some current algorithms for the SW problem, we propose a recursive bi-partitioning heuristic that performs well. Then a distributed framework is proposed, which allows us to parallelize these algorithms by distributing the nodes to some local machines. We test our proposed methods on synthetic and real-world datasets and show that these methods are near-optimal and outperform the state-of-the-art algorithm(SIP).

8.2. Future Work

Here we list possible directions for future work□

- Under our current assumption, the graphs dealt with are static. However, in most real-world applications, graphs are usually time-varying. Therefore, how to update the partitioning scheme accordingly needs plenty of work.
- We assume that graph signals are stationary. This cannot but cannot be over a long time duration. Therefore, some online change detection algorithms can be developed.
- We have derived a general lower bound for the distributed partitioning framework. However, it is not as tight enough for our proposed algorithms to observe the results. So deriving a tighter bound can also be a possible subject of future work.

References

[1] David I Shuman et al. “The emerging field of signal processing on graphs: Extending high-dimensional data analysis to networks and other irregular domains”. In: *IEEE signal processing magazine* 30.3 (2013), pp. 83–98.

[2] Siheng Chen et al. “Signal denoising on graphs via graph filtering”. In: *2014 IEEE Global Conference on Signal and Information Processing (GlobalSIP)*. IEEE. 2014, pp. 872–876.

[3] Siheng Chen et al. “Discrete Signal Processing on Graphs: Sampling Theory”. In: *IEEE Transactions on Signal Processing* 63.24 (2015), pp. 6510–6523. DOI: 10.1109/TSP.2015.2469645.

[4] Aamir Anis, Akshay Gadde, and Antonio Ortega. “Towards a sampling theorem for signals on arbitrary graphs”. In: *2014 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE. 2014, pp. 3864–3868.

[5] Siheng Chen et al. “Signal recovery on graphs: Variation minimization”. In: *IEEE Transactions on Signal Processing* 63.17 (2015), pp. 4609–4624.

[6] Alberto Natali et al. “Learning time-varying graphs from online data”. In: *IEEE Open Journal of Signal Processing* 3 (2022), pp. 212–228.

[7] Zhao Kang et al. “Robust graph learning from noisy data”. In: *IEEE transactions on cybernetics* 50.5 (2019), pp. 1833–1843.

[8] Patric Hagmann et al. “Mapping the structural core of human cerebral cortex”. In: *PLoS biology* 6.7 (2008), e159.

[9] Daniel A Spielman. “Spectral graph theory and its applications”. In: *48th Annual IEEE Symposium on Foundations of Computer Science (FOCS’07)*. IEEE. 2007, pp. 29–38.

[10] Andrew Ng, Michael Jordan, and Yair Weiss. “On spectral clustering: Analysis and an algorithm”. In: *Advances in neural information processing systems* 14 (2001).

[11] Douglas Brent West et al. *Introduction to graph theory*. Vol. 2. Prentice hall Upper Saddle River, 2001.

[12] Aliaksei Sandryhaila and Jose MF Moura. “Discrete signal processing on graphs: Frequency analysis”. In: *IEEE Transactions on Signal Processing* 62.12 (2014), pp. 3042–3054.

[13] Leo J Grady and Jonathan R Polimeni. *Discrete calculus: Applied analysis on graphs for computational science*. Vol. 3. Springer, 2010.

[14] Nathanaël Perraudin et al. “GSPBOX: A toolbox for signal processing on graphs”. In: *arXiv preprint arXiv:1408.5781* (2014).

[15] Ulrike Von Luxburg. “A tutorial on spectral clustering”. In: *Statistics and computing* 17.4 (2007), pp. 395–416.

[16] Yanbin He. *Adaptive graph partition methods for structured graphs*. Jan. 2021. URL: http://resolver.tudelft.nl/uuid:f9314ad7-07ee-4aaf-850d-04d03233ca00.

[17] Aristidis Likas, Nikos Vlassis, and Jakob J Verbeek. “The global k-means clustering algorithm”. In: *Pattern recognition* 36.2 (2003), pp. 451–461.

[18] Alan V Oppenheim et al. *Signals & systems*. Pearson Educación, 1997.

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

[20] Han Shomorony and A Salman Avestimehr. “Sampling large data on graphs”. In: *2014 IEEE Global Conference on Signal and Information Processing (GlobalSIP)*. IEEE. 2014, pp. 933–936.

[21] Antonio G Marques et al. “Sampling of graph signals with successive local aggregations”. In: *IEEE Transactions on Signal Processing* 64.7 (2015), pp. 1832–1843.

[22] Sunil K Narang, Akshay Gadde, and Antonio Ortega. “Signal processing techniques for interpolation in graph structured data”. In: *2013 IEEE International Conference on Acoustics, Speech and Signal Processing*. IEEE. 2013, pp. 5445–5449.

[23] Abhimanyu Das and David Kempe. “Algorithms for subset selection in linear regression”. In: *Proceedings of the fortieth annual ACM symposium on Theory of computing*. 2008, pp. 45–54.

[24] Luiz FO Chamon and Alejandro Ribeiro. “Greedy sampling of graph signals”. In: *IEEE Transactions on Signal Processing* 66.1 (2017), pp. 34–47.

[25] Siddharth Joshi and Stephen Boyd. “Sensor selection via convex optimization”. In: *IEEE Transactions on Signal Processing* 57.2 (2008), pp. 451–462.

[26] Satoru Fujishige. *Submodular functions and optimization*. Elsevier, 2005.

[27] Ming-Yu Liu et al. “Entropy-rate clustering: Cluster analysis via maximizing a submodular function subject to a matroid constraint”. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 36.1 (2013), pp. 99–112.

[28] Andreas Krause and Daniel Golovin. “Submodular function maximization.” In: *Tractability* 3 (2014), pp. 71–104.

[29] Francis Bach et al. “Learning with submodular functions: A convex optimization perspective”. In: *Foundations and Trends® in Machine Learning* 6.2-3 (2013), pp. 145–373.

[30] Satoru Iwata. “Submodular function minimization”. In: *Mathematical Programming* 112.1 (2008), pp. 45–64.

[31] Niv Buchbinder et al. “Submodular maximization with cardinality constraints”. In: *Proceedings of the twenty-fifth annual ACM-SIAM symposium on Discrete algorithms*. SIAM. 2014, pp. 1433–1452.

[32] George L Nemhauser, Laurence A Wolsey, and Marshall L Fisher. “An analysis of approximations for maximizing submodular set functions—I”. In: *Mathematical programming* 14.1 (1978), pp. 265–294.

[33] Nathanaël Perraudin and Pierre Vandergheynst. “Stationary signal processing on graphs”. In: *IEEE Transactions on Signal Processing* 65.13 (2017), pp. 3462–3477.

[34] Benjamin Girault. “Stationary graph signals using an isometric graph translation”. In: *2015 23rd European Signal Processing Conference (EUSIPCO)*. IEEE. 2015, pp. 1516–1520.

[35] Antonio G Marques et al. “Stationary graph processes and spectral estimation”. In: *IEEE Transactions on Signal Processing* 65.22 (2017), pp. 5911–5926.

[36] Stephen Boyd, Stephen P Boyd, and Lieven Vandenberghe. *Convex optimization*. Cambridge university press, 2004.

[37] Luiz FO Chamon and Alejandro Ribeiro. “Universal bounds for the sampling of graph signals”. In: *2017 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE. 2017, pp. 3899–3903.

[38] Thomas Kailath, Ali H Sayed, and Babak Hassibi. *Linear estimation*. BOOK. Prentice Hall, 2000.

[39] Cristian Rusu and John Thompson. “Node sampling by partitioning on graphs via convex optimization”. In: *2017 Sensor Signal Processing for Defence Conference (SSPD)*. IEEE. 2017, pp. 1–5.

[40] Steven M Kay. *Fundamentals of statistical signal processing: estimation theory*. Prentice-Hall, Inc., 1993.

[41] Manohar Shamaiah, Siddhartha Banerjee, and Haris Vikalo. “Greedy sensor selection: Leveraging submodularity”. In: *49th IEEE conference on decision and control (CDC)*. IEEE. 2010, pp. 2572–2577.

[42] Roshni Chakraborty et al. “Finding Representative Sampling Subsets in Sensor Graphs using Time Series Similarities”. In: *arXiv preprint arXiv:2202.08504* (2022).

[43] Siheng Chen, Aliaksei Sandryhaila, and Jelena Kovačević. “Sampling theory for graph signals”. In: *2015 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE. 2015, pp. 3392–3396.

[44] Josefine Holm et al. “Lifetime maximization of an internet of things (iot) network based on graph signal processing”. In: *IEEE Communications Letters* 25.8 (2021), pp. 2763–2767.

[45] Emmanuel J Candes, Michael B Wakin, and Stephen P Boyd. “Enhancing sparsity by reweighted ℓ 1 minimization”. In: *Journal of Fourier analysis and applications* 14.5 (2008), pp. 877–905.

[46] Zheng Zhang et al. “A survey of sparse representation: algorithms and applications”. In: *IEEE access* 3 (2015), pp. 490–530.

[47] Shahar Dobzinski, Noam Nisan, and Michael Schapira. “Approximation algorithms for combinatorial auctions with complement-free bidders”. In: *Proceedings of the thirty-seventh annual ACM symposium on Theory of computing*. 2005, pp. 610–618.

[48] Shahar Dobzinski and Michael Schapira. “An improved approximation algorithm for combinatorial auctions with submodular bidders”. In: *Proceedings of the seventeenth annual ACM-SIAM symposium on Discrete algorithm*. 2006, pp. 1064–1073.

[49] Uriel Feige. “On maximizing welfare when utility functions are subadditive”. In: *Proceedings of the thirty-eighth annual ACM symposium on Theory of computing*. 2006, pp. 41–50.

[50] Benny Lehmann, Daniel Lehmann, and Noam Nisan. “Combinatorial auctions with decreasing marginal utilities”. In: *Proceedings of the 3rd ACM conference on Electronic Commerce*. 2001, pp. 18–28.

[51] Niv Buchbinder et al. “A tight linear time (1/2)-approximation for unconstrained submodular maximization”. In: *SIAM Journal on Computing* 44.5 (2015), pp. 1384–1402.

[52] Moran Feldman. “Maximizing symmetric submodular functions”. In: *ACM Transactions on Algorithms (TALG)* 13.3 (2017), pp. 1–36.

[53] Subhash Khot et al. “Inapproximability results for combinatorial auctions with submodular utility functions”. In: *Internet and Network Economics: First International Workshop, WINE 2005, Hong Kong, China, December 15-17, 2005. Proceedings 1*. Springer. 2005, pp. 92–101.

[54] Uriel Feige and Jan Vondrák. “The submodular welfare problem with demand queries”. In: *Theory of Computing* 6.1 (2010), pp. 247–290.

[55] Baharan Mirzasoleiman et al. “Distributed submodular maximization: Identifying representative elements in massive data”. In: *Advances in Neural Information Processing Systems* 26 (2013).

[56] Rafael da Ponte Barbosa et al. “A new framework for distributed submodular maximization”. In: *2016 IEEE 57th Annual Symposium on Foundations of Computer Science (FOCS)*. Ieee. 2016, pp. 645–654.

[57] Baharan Mirzasoleiman et al. “Distributed submodular maximization”. In: *The Journal of Machine Learning Research* 17.1 (2016), pp. 8330–8373.

[58] Jeffrey Dean and Sanjay Ghemawat. “MapReduce: simplified data processing on large clusters”. In: *Communications of the ACM* 51.1 (2008), pp. 107–113.

[59] Cheng-Tao Chu et al. “Map-reduce for machine learning on multicore”. In: *Advances in neural information processing systems* 19 (2006).

[60] Jaliya Ekanayake, Shrideep Pallickara, and Geoffrey Fox. “Mapreduce for data intensive scientific analyses”. In: *2008 IEEE Fourth International Conference on eScience*. IEEE. 2008, pp. 277–284.